UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Exploring Neuroevolution Fitness Landscapes for Optimization and Generalization

Nuno Miguel Vasconcelos Rodrigues

Mestrado em Engenharia Informática
Especialização em Interação e Conhecimento

Dissertação orientada por:
Prof.ª Doutora Sara Guilherme Oliveira da Silva
Prof.º Doutor Leonardo Vanneschi

2020

Acknowledgments

First, I would like to greatly thank Prof. Sara Silva and Prof. Leonardo Vanneschi, for helping me and guiding me for the past years. The only reason I am academically able to be where I am now is because they trusted me and gave me a chance.

I would also like to thank my parents, grandmother, and Mariana for their unconditional support during both happy and though times.

Lastly, regarding personal acknowledgments, I would like to thank the following people for their friendship and help over the years: Ana Tam, João Batista, João Loureiro, Mariana Casanova, Pedro Vieira, Pedro Andrade, Sofia, and multiple others from the groups Fculianos and Químicas.

Finally, I would like to thank the following institutions and projects: FCT, Portugal, through funding of LASIGE Research Unit (UIDB/00408/2020, UIDP/00408/2020) and projects BINDER (PTDC/CCI-INF/29168/2017), GADgET (DSAIPA/DS/0022/2018), AICE (DSAIPA/DS/0113/2019), and PREDICT (PTDC/CCI-CIF/29877/2017).

Resumo

Paisagens de aptidão (*fitness landscapes*) são um conceito útil e largamente investigado para estudar as dinâmicas de meta-heurísticas. Nas últimas duas décadas têm sido utilizadas com sucesso para estimar as capacidades de otimização de diversos tipos de algoritmos evolutivos, tal como algoritmos genéticos e programação genética. No entanto, até à data nunca foram utilizadas para estudar o desempenho de algoritmos de aprendizagem automática em dados nunca vistos durante o treino, e nunca foram aplicadas para estudar as paisagens geradas por neuroevolução. Coincidentemente, apesar de já existir há quase três décadas e ainda ser uma área de investigação com um crescimento rápido e dinâmico, a neuroevolução ainda tem falta de fundações teóricas e metodológicas, fundações essas que podem ser dadas através da aplicação de paisagens de aptidão. Esta dissertação tem como objetivo preencher estas lacunas ao aplicar paisagens de aptidão à neuroevolução, usando este conceito para inferir informação útil sobre a capacidade de aprendizagem e generalização deste método de aprendizagem automática. De forma a realizar esta tarefa, desenvolvemos e usámos um algoritmo de neuroevolução baseado em gramáticas que gera redes neuronais convolucionais, e estudámos a dinâmica de três operadores de mutação distintos usados para evoluir múltiplos aspetos das redes neuronais. De forma a caracterizar as paisagens de aptidão, estudámos a autocorrelação (*autocorrelation*), medida entrópica de rugosidade (*entropic measure of ruggedness*), nuvens de aptidão (*fitness clouds*), medidas de gradiente (*gradient measures*) e o coeficiente de declive negativo (*negative slope coefficient*), e ao mesmo tempo discutimos porque é que apesar de não usarmos outras medidas, tais como redes de ótimos locais (*local optima networks*) e correlação aptidão distância (*fitness distance correlation*), estas podem providenciar resultados interessantes. Também propomos o uso de duas novas medidas de avaliação: nuvens de densidade, uma nova medida desenvolvida nesta tese com capacidade de dar informação visual sobre a distribuição de amostras, e a medida de sobreajustamento (*overfitting*), que é derivada de uma medida já existente e usada em programação genética. Os resultados demonstram que as medidas usadas são apropriadas e produzem resultados precisos no que toca a estimar tanto a capacidade de aprendizagem como a habilidade de generalização das configuração de neuroevolução consideradas.

Palavras-chave: Paisagens de Fitness, Neuroevolução, Redes Neuronais Convolucionais, Generalização

Abstract

Fitness landscapes are a useful and widely investigated concept for studying the dynamics of meta-heuristics. In the last two decades, they have been successfully used for estimating the optimization capabilities of different flavors of evolutionary algorithms, including genetic algorithms and genetic programming. However, so far they have not been used for studying the performance of Machine Learning (ML) algorithms on unseen data, and they have not been applied to study neuroevolution landscapes. Coincidentally, despite having existed for almost three decades and still being a dynamic and rapidly growing research field, neuroevolution still lacks theoretical and methodological foundations, which could be provided by the application of fitness landscapes. This thesis aims to fill these gaps by applying fitness landscapes to neuroevolution, using this concept to infer useful information about the learning and generalization ability of the ML method. For this task, we developed and used a grammar-based neuroevolution approach to generate convolutional neural networks, and studied the dynamics of three different mutation operators used to evolve multiple aspects of the networks. To characterize fitness landscapes, we studied autocorrelation, entropic measure of ruggedness, fitness clouds, gradient measures and negative slope coefficient, while also discussing why other measures such as local optima networks and fitness distance correlation, despite not being used, could provide interesting results. Also, we propose the use of two additional evaluation measures: density clouds, a new measure developed in this thesis that can provide visual information regarding the distribution of samples, and overfitting measure, which is derived from a measure used in genetic programming. The results show that the used measures are appropriate and produce accurate results when estimating both the learning capability and the generalization ability of the considered neuroevolution configurations.

Keywords: Fitness Landscapes, Neuroevolution, Convolutional Neural Networks, Generalization

Resumo alargado

Paisagens de aptidão (*fitness landscapes*) é um conceito derivado da metáfora de paisagem originada em genética, e podem ser vistas como um gráfico em que os pontos no eixo horizontal representam diferentes genótipos de indivíduos num espaço de procura, enquanto os pontos no eixo vertical representam a aptidão de cada um desses indivíduos. Este mapeamento, que visa exprimir relações de vizinhança entre diversos indivíduos, bem como a relação de aptidão entre os mesmos, produzindo um gráfico com um conjunto de vales e colinas, onde a tarefa de encontrar a melhor solução corresponde a encontrar o pico mais alto ou mais baixo, dependendo se é um problema de maximização ou minimização. Esta metáfora é útil no que toca a perceber a dificuldade dos problemas. No entanto, em termos práticos, uma paisagem de aptidão é impossível de visualizar, devido ao vasto tamanho do espaço de procura e ao facto da vizinhança ser multidimensional. Por esta razão, ao longo das duas últimas décadas, vários investigadores têm desenvolvido várias medidas matemáticas de forma a capturar características das paisagens e exprimi-las numericamente. Apesar de terem sido extensivamente aplicadas e exploradas em diversos tipos de algoritmos evolutivos, tal como algoritmos genéticos e programação genética, nunca foram analizadas paisagens de aptidão com o objetivo de estudar o desempenho de algoritmos de aprendizagem automática em dados nunca vistos durante o treino, nem nunca foram aplicadas em neuroevolução.

Neuroevolução é um ramo da computação evolutiva onde são usados algoritmos genéticos para evoluir a topologia, pesos, parâmetros e estratégias de aprendizagem de redes neuronais. Apesar de já existir há mais de três décadas e ter aplicações em inúmeras áreas desde visão computacional até aprendizagem por reforço, a neuroevolução ainda tem falta de fundações teóricas e metodológicas, fundações essas que podem ser dadas através da aplicação de paisagens de aptidão.

Esta dissertação tem como objetivo preencher estas lacunas ao aplicar paisagens de aptidão à neuroevolução, usando este conceito de forma a inferir informação útil sobre a capacidade de aprendizagem e generalização deste método de aprendizagem automática.

A metodologia que seguimos durante a realização desta dissertação foi composta por quatro diferentes etapas: desenvolvimento de um algoritmo de neuroevolução baseado em gramáticas; escolha das medidas de análise de paisagens de aptidão a utilizar e desenvolvimento de duas novas medidas; escolha dos problemas a utilizar e criação de um

novo conjunto de dados sintético propenso a sobreajustamento (*overfitting*); análise meticulosa dos resultados obtidos. Em termos de arquitetura das redes escolhemos usar redes neuronais convolucionais. Decidimos que a codificação ia ser composta por seis genes diferentes, cinco deles para camadas, sendo estas as seguintes: *Dense*, *Dropout*, *Convolutional*, *Pooling*, *Flatten*, e um para o optimizador. O gene *Flatten* não se encontra expresso na gramática pois não tem qualquer parâmetro e o seu único uso é transformar o output multidimensional proveniente das camadas de convolução para um input unidimensional, de forma a ser usado pelas camadas densas. Devido à elevada quantidade de possíveis mutações, restringimos o nosso estudo apenas a três: mutações de aprendizagem, que alteram os parâmetros relativos ao optimizador; mutações de parâmetros, que alteram os valores dos parâmetros das diferentes camadas; alterações topológicas, que adicionam ou removem camadas. Quanto à avaliação das redes, usamos como função de erro a *Sparse Categorical Cross-Entropy*, dado que os problemas continham várias classes e a codificação usada não era a *One-Hot Encoding*. Escolhemos como optimizador o *Stochastic Gradient Descent* devido ao facto de já existirem estudos sobre os possíveis valores que os seus parâmetros podem assumir, facilitando a discretização na gramática, e porque generaliza melhor do que outros optimizadores mais usados como o *ADAM*. Durante o processo evolutivo foi usado o erro em treino como função de aptidão.

Na segunda etapa, começámos por escolher seis medidas já existentes de análise de paisagens de aptidão: Autocorrelação (*Autocorrelation*), medida entrópica de rugosidade (*Entropic Measure of Ruggedness*), nuvens de aptidão (*Fitness Clouds*), medidas de gradiente (*Gradient measures*), coeficiente de decline negativo (*Negative Slope Coefficient*). Começando pela autocorrelação, medida entrópica de rugosidade e medidas de gradiente, foi necessário realizar passeios pela paisagem, e optamos por realizar passeios seletivos (*selective walks*) em vez dos mais comuns passeios aleatórios (*random walks*), dado que estes aplicam um pouco de pressão seletiva, restringindo assim o espaço de procura. Para as nuvens de aptidão e coeficiente de declive negativo foi necessário criar uma amostra e gerar uma vizinhança para cada um dos pontos da amostra. Para fazer estas amostras recorremos à técnica *Metropolis-Hastings* dado que este método aplica pressão seletiva e consegue assim imitar o comportamento de um algoritmo de procura, reduzindo o número de pontos redundantes. Seguidamente, utilizámos duas novas medidas de análise de paisagens: nuvens de densidade (*density clouds*) e medida de sobreajustamento (*Overfitting Measure*). As nuvens de densidade têm como objetivo visualizar a distribuição da densidade das nuvens de aptidão, e são geradas aplicando um kernel gaussiano à mesma amostra que gera as nuvens de aptidão. Esta medida foi desenvolvida como alternativa a uma outra medida chamada densidade de estados (*Density of States*) pois esta, apesar de produzir valores concretos de densidade, requer uma amostra muito maior e diferente, porque é necessário encontrar os vizinhos de cada ponto da amostra em vez de os gerar, uma tarefa extremamente difícil em neuroevolução devido ao enorme

espaço de procura originado pela elevada quantidade de parâmetros das redes. A medida de sobreajustamento é uma medida que tem como objetivo detetar sobreajustamento nos passeios pela paisagem de aptidão, e é calculada através da comparação entre os melhores pontos e os restantes, pelo que foram utilizados os mesmo passeios usados para calcular a autocorrelação e a medida entrópica de rugosidade. Esta medida foi adaptada de uma medida já existente usada para o mesmo fim, mas aplicada à analise da evolução em programação genética.

Passando para a terceira etapa, escolhemos quatro problemas bem conhecidos com diferentes graus de dificuldade: *MNIST*, *Fashion-MNIST*, *CIFAR10* e *SVHN*. Nenhum destes problemas é propicio à ocorrência de sobreajustamento, apesar de ser fácil obter maus resultados, pelo que criámos um novo conjunto de dados baseado no *MNIST*, a que chamamos de *Small and Mislabelled*, composto pelos últimos 30% do *MNIST* onde trocámos metade das etiquetas ímpares. De forma a não introduzir nenhum enviesamento, não foi feito qualquer tipo de pré-processamento ou aumento de dados, apenas foi ajustada a escala das imagems de forma a colocar os valores dos pixeis no intervalo entre $[0; 1]$.

Ao analisar os resultados obtidos nestes cinco problemas, obtemos duas confirmações: os operadores de mutação de aprendizagem e parâmetros são capazes de evoluir facilmente as redes obtendo resultados bastante bons, no entanto, o operador de mutações topológicas tem mais dificuldade e acaba por gerar paisagens bastante irregulares; as medidas aplicadas, à exceção do coeficiente de declive negativo, são bons indicadores do grau de dificuldade, tanto em treino como em teste, e são também bons indicadores da capacidade de generalização pois conseguem facilmente indentificar sobreajustamento. Quanto ao coeficiente de declive negativo precisaríamos de uma amostra de pontos bastante superior, o que não nos foi possível devido a constrangimentos computacionais, de modo a obter resultados precisos e fiáveis.

No futuro pretendemos expandir esta análise a mais medidas, tais como redes de ótimos locais (*Local Optima Networks*) e correlação aptidão distância (*fitness distance correlation*), em mais problemas de forma a desenvolver mais mecanismos bem estabelecidos e com boa motivação teórica que sejam capazes de simplificar significativamente as fases de configuração e ajuste de parâmetros da neuroevolução. Pretendemos também continuar a expandir o algoritmo de neuroevolução para que possa integrar mais camadas, operadores, e parâmetros evolutivos, de forma a tornar-se mais competitivo. Por fim, pretendemos analisar em mais detalhe os resultados obtidos pelo operador das mutações topológicas, pois suspeitamos que os resultados fracos que ele origina se devem à degradação da propagação do erro originado pelo empilhamento de camadas.

Palavras-chave: Paisagens de Fitness, Neuroevolução, Redes Neuronais Convolucionais, Generalização

Contents

List of Figures

List of Tables

Acronyms

CNN Convolutional Neural Network.

EA Evolutionary Algorithm.

EMR Entropic Measure of Ruggedness.

FL Fitness Landscape.

GA Genetic Algorithm.

ML Machine Learning.

NSC Negative Slope Coefficient.

OFM Overfitting Measure.

SGD Stochastic Gradient Descent.

Chapter 1

Introduction

The concept of fitness landscapes (FLs) (Wright (1932); Stadler (2002)) has been used many times to characterize the dynamics of meta-heuristics in optimization tasks. Particularly, several measures have been developed with the goal of capturing essential characteristics of FLs that can provide useful information regarding the difficulty of different optimization problems. Among these measures, autocorrelation (Weinberger (1990)), gradient measures, fitness clouds (Verel et al. (2003); Vanneschi et al. (2006)), negative slope coefficient (NSC) (Vanneschi et al. (2006)) and entropic measure of ruggedness (EMR) (Vassilev (2000); Vassilev et al. (2000, 2003)) have been intensively studied, revealing to be useful indicators of the ruggedness and difficulty of the FLs induced by several variants of local search meta-heuristics and evolutionary algorithms (EAs) (Verel (2012)). However, to the best of our knowledge, despite the existence of a few works on FLs applied to standard neural networks (Rakitianskaia et al. (2016); Gallagher (2001)), no measure related to FLs has ever been used for studying the performance of machine learning (ML) algorithms on unseen data, nor for characterizing the dynamics of neuroevolution. In this work, we adapt the well-known definitions of autocorrelation, gradient measures, fitness clouds, NSC and EMR to neuroevolution. We introduce two novel measures, called density clouds and overfitting measure, not only to study the optimization effectiveness of various neuroevolution configurations but also to characterize their performance on unseen data. We also introduce our own grammar-based neuroevolution approach, inspired by existing systems.

Neuroevolution is a branch of evolutionary computation that has been around for almost three decades, with application in multiple areas such as supervised classification tasks (Gómez et al. (2018)) and agent building (Stanley and Miikkulainen (2002)). In neuroevolution, an EA is used to evolve weights, topologies and/or hyper-parameters of artificial neural networks. In this study, we focus on the evolution of convolutional neural networks (CNNs), because they are one of the most popular deep neural network architectures, and yet they have a vast amount of tunable parameters that are difficult to set, which makes them perfect for testing the capabilities of neuroevolution. For testing the reliabil-

ity of the studied measures in predicting the performance of neuroevolution of CNNs on training and unseen data, we consider three different types of mutations and four different multiclass classification problems, with different degrees of known difficulty. For each type of mutation, and for each of the studied problems, we calculate the value of these measures and we compare them to the results obtained by actual simulations performed with our neuroevolution system.

We consider this work as the first proof of concept in a wider study, aimed at establishing the use of measures to characterize neuroevolution of CNNs. If successful, this study will be extremely impactful. CNNs normally have a slow learning phase, which makes neuroevolution a very intensive computational process, as it requires the evaluation of several CNNs in each generation. For this reason, the task of executing simulations to choose among different types of genetic operators, and/or among several possible parameter settings, is normally prohibitively expensive. This may result in the adoption of suboptimal configurations that not only will require a tremendous amount of training time due to learning difficulty, but also will perform poorly after training because of poor generalization ability. On the other hand, the calculation of the studied measures is much faster, and therefore it can help us find appropriate neuroevolution configurations much more efficiently, with one such example being presented in Section 4. The result may be that the time spent on calculating the measures will be largely compensated with optimized configurations that will learn faster and generalize better. Computational constraints have also prompted us to choose simple measures that are both quick to calculate and, as our results demonstrate, effective.

1.1 Contributions

The main contributions of this dissertation are the following:

1. Studying for the first time the application of Fitness Landscapes to Neuroevolution. This contribution was accepted (Rodrigues et al. (2020)) as a full paper in the core B conference IEEE Congress on Evolutionary Computation (CEC) and will be presented in 19-24th July, 2020;

2. Expanding the previous analysis of Neuroevolution by application of more Fitness Landscape measures and the study of its Generalization capabilities. This contribution was published (Rodrigues et al. (2020)) in the Q1 journal IEEE Access;

3. Development of a new Fitness Landscape analysis method named Density Clouds, that gives visual information regarding the density of samples (Rodrigues et al. (2020));

4. Development of a novel Fitness Landscape analysis method named Overfitting Measure, that can identify overfitting in walks through the landscape, something that has never been done before (Rodrigues et al. (2020));

5. Development of TFNE, a modular, Tensorflow built, grammar-based Neuroevolution package with Fitness Landscape analysis methods (Rodrigues et al. (2020)) (https://github.com/NMVRodrigues/TFNE).

1.2 Document Structure

This document is organized as follows. In Section 2, we introduce all the concepts we will use throughout this work, including neuroevolution, FL, and the used measures. Section 3 goes through the methodology we followed, from creating the neuroevolution algorithm to the development of new FL analysis measures, finishing with the experimental setup. In Section 4, we present the results we obtained as well as their detailed analysis. Finally, Section 5 concludes the paper and suggests ideas for future research.

Chapter 2

Concepts

2.1 Convolutional Neural Networks

Convolutional neural networks (CNN) were originally introduced by Lecun et al. (1998) and inspired by Fukushima's NeoCognitron (Fukushima (1980)). Over the years, CNNs have demonstrated their preeminence in multiple areas such as computer vision, human activity recognition (Tsironi et al. (2017); Ordóñez et al. (2016)) and visual recognition tasks, such as image classification and semantic segmentation (Yu and Koltun (2015)). The structure of a CNN is composed of three main operations: convolution operations, pooling operations and fully connected layers.

In convolutional layers, a set of kernels, also called filters, with a predefined size, set of weights, padding and stride, travel from left to right, top to bottom, over the image, producing a feature map. These kernels can be either two dimensional or three dimensional, depending on if the image they are iterating over is either greyscale or RGB, respectively.

Pooling layers often follow convolutional layers. They function similarly to convolution operations, where a kernel of a predefined size, stride and padding iterates over its input. The main difference is that in pooling operations, this kernel iterates over feature maps, to reduce dimensionality, by downsampling each feature map independently, reducing their two-dimensional size, without any change to the depth. Among the multiple types of pooling operations, the most common ones are max-pooling and average-pooling.

After the feature maps are extracted from the images, they are flattened, reshaped into one-dimensional arrays, to be fed as input to the fully connected/dense layers which perform the classification. During this last section it is not imperative to only have fully connected/dense layers. There are a wide variety of other layers and units that can be used, such as long short-term memory units (Hochreiter and Schmidhuber (1997)) and gate recurrent units (Cho et al. (2014)), all to name a few.

2.2 Genetic algorithms

Genetic algorithms (GAs) are a biologically inspired subfield within evolutionary computation (Eiben and Smith (2015)) created by Holland (1992) and based on the theory of evolution and natural selection proposed by Darwin (1859).

In GAs, for a given problem, a population of solutions, commonly named individuals, is created and iterated through an evolutionary cycle (Figure 2.1). During this cycle, each individual is first evaluated according to a fitness function whose output gives an indication of how well that solution performs. After being evaluated, individuals are selected for reproduction to create a population of offspring by application of genetic operations such as mutation or crossover. This cycle goes on for multiple generations and can be stopped by either reaching the generation limit or by multiple other stop conditions.

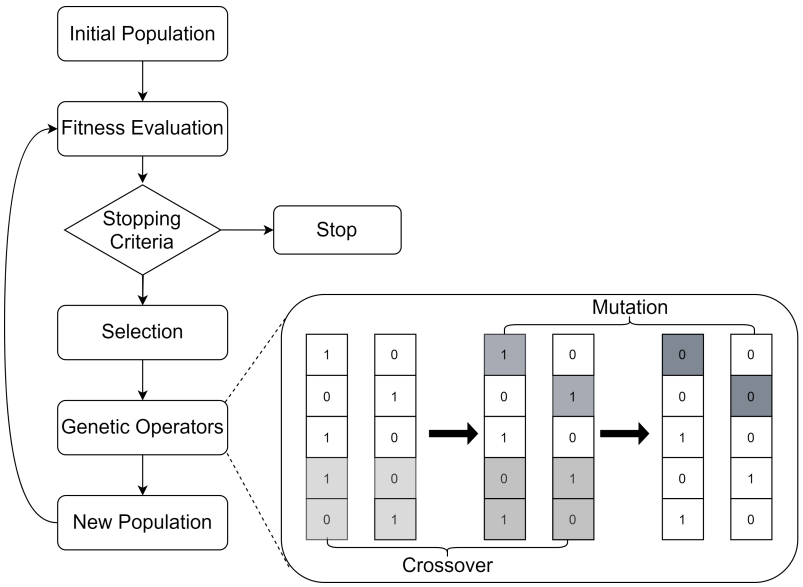


Figure 2.1: A standard evolutionary cycle for genetic algorithms along with a representation of the crossover and mutation genetic operators

For selection mechanisms, one of the most commonly used is the Tournament selection, where a sample of N parents are taken at random from the population, and the fittest one from that sample is selected for breeding.

Regarding the genetic operators, the two most commonly used are the crossover and mutation operators. Crossover takes two parents, selects a crossover point in each parent and swaps all genetic material after that point, creating two offspring. Mutation uses one parent and, selects a mutation point and changes the genetic material after that point, creating one offspring.

2.3 Neuroevolution

Neuroevolution is a subfield within evolutionary algorithms (Fogel (1962); Eiben and Smith (2015)) in which, normally, a genetic algorithm is used to evolve neural networks. This allows for an exhaustive search through the structural (topology and weights) space with the aid of a genetic operator. Apart from saving the time required to decide on a structure (Gruau et al. (1996)), neuroevolution reduces the human bias involved in the process.

The first neuroevolution algorithms only evolved the weights of the networks (Miller et al. (1989)), but now these algorithms have been extended to also evolve both the parameters and the topology of the networks (Angeline et al. (1994); Chen et al. (1993); Branke (1995); Gruau et al. (1996); Xin Yao (1999)).

In neuroevolution, during the evolutionary process, each individual in the population represents the encoded genotype of a network. The encoding of the genotype is a crucial step that determines what kinds of genetic operations can be applied to the individuals. The most common encoding strategies are briefly explained in 2.3.1.

Some of the most noteworthy neuroevolution algorithms developed to date include EPNet (Yao and Liu (1997)), NEAT (Stanley and Miikkulainen (2002)), EANT (Kassahun and Sommer (2005)), and hyperNEAT (Stanley et al. (2009)).

2.3.1 Encodings

Direct Encoding

In direct encoding strategies, the components of the networks have a 1-to-1 mapping with the encoded genes. This sort of encoding makes it easier to analyze the genome and understand how the network is built. A downside, however, is that the larger the network, the larger the encoded genome, which may make analyzing the genome a daunting task. A prime example of such an encoding is the one used in NEAT (Stanley and Miikkulainen (2002)), shown in Figure 2.2:

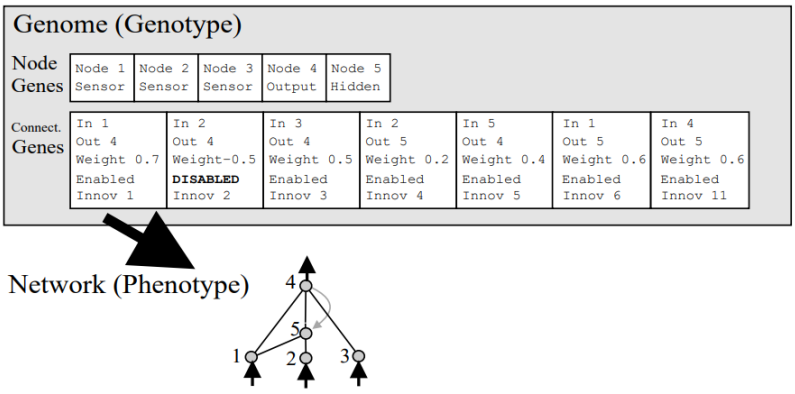


Figure 2.2: Direct encoding example used in NEAT

Indirect encoding

In indirect encoding strategies, the genotype can be viewed as a compressed representation of the network, where there is no direct mapping from genes to the components of the network. This representation allows for the exploration of patterns and regularities within the genome. A prime example of such encoding is the CPPN (Stanley (2007)).

Developmental encoding

In developmental encoding strategies, it is assumed that networks are modular and composed of small building blocks for specific tasks, that when decoded and merged produce the full network, similar to how brains work according to Gruau (1994). A prime example of such encoding can be found in the work of Kitano (1990) where he used a grammar system with evolvable rules to recursively develop the matrix that defined the networks connections.

Grammatical encoding

Grammatical encoding strategies are mainly used when performing Grammatical evolution (Ryan et al. (1998)). In this strategy, the information needed to generate the individuals is encoded in a formal grammar. This promotes extreme flexibility, as it is possible to generate multiple types of networks with the same grammar (**e.g.** CNNs or Autoencoders), and since all information is discriminated it is very simple to perform changes.

Some successful works using this encoding strategy include Gómez et al. (2018) and Assunção et al. (2018).

2.3.2 NeuroEvolution of Convolutional Neural Networks

Only recently, due to hardware advances, has neuroevolution been applied to deep network architectures such as CNNs.

The first works using neuroevolution with CNNs were done with fixed topology networks. Koutník et al. (2014) evolved the weights of a fixed topology network and Young et al. (2015) evolved a set of parameters (kernel size and the number of kernels) from a fixed topology network. Loshchilov and Hutter (2016) followed a similar logic, evolving 19 different hyperparameters, ranging from the learning algorithm to the network structure on a network with a fixed number of layers and not topology.

Some NEAT and hyperNEAT variations were also proposed (Verbancsics and Harguess (2015); Miikkulainen et al. (2019)), both producing average results.

Two key works were the ones by Xie and Yuille (2017) and Gómez et al. (2018). Xie and Yuille (2017) proposed Genetic CNN, a fixed-length binary encoding based algorithm that evolved both the hyperparameters and topology of convolutional and pooling layers, surpassing VGGNet (Simonyan and Zisserman (2015)) which was one of the current

state-of-the-art models. Gómez et al. (2018) published a very in-depth work regarding evolutionary CNNs. In this work, all aspects of a CNN were evolved, from hyperparameters to learning parameters and topology, on all the constituting layers, using GAs and grammatical evolution, obtaining results better than some state of the art models.

Most recently there have been multiple works aimed at facilitating (Assunção et al. (2018)), optimizing (Assunção et al. (2019)) and applying incremental development (Assunção et al. (2020)) to neuroevolution of CNNs.

2.4 Fitness Landscapes

Using a landscape metaphor to gain insight about the workings of a complex system originates with the work of Wright on genetics (Wright (1932)). Probably, the simplest definition of FL is the following one: a FL can be seen as a plot where the points in the horizontal direction represent the different individual genotypes in a search space and the points in the vertical direction represent the fitness of each one of these individuals (Langdon and Poli (2002)). If genotypes can be visualized in two dimensions, the plot can be seen as a three-dimensional "map", which may contain peaks and valleys. The task of finding the best solution to the problem is equivalent to finding the highest peak (for maximization problems) or the lowest valley (for minimization). The problem solver is seen as a short-sighted explorer searching for those optimal spots. Crucial to the concept of FL is that solutions should be arranged in a way that is consistent with a given neighborhood relationship. Indeed, a FL is completely defined by the triple:

$$(S, f, N)$$

where S is the set of all admissible solutions (the search space), f is the fitness function, a function used to measure the quality of the solutions found by the algorithm, and, N is the neighborhood. Generally, the neighborhood should have a relationship with the transformation (mutation) operator used to explore the search space. A typical example is to consider as neighbors two solutions a and b if and only if b can be obtained by applying mutation to a.

The FL metaphor can be helpful for understanding the difficulty of a problem, i.e., the ability of a searcher to find the optimal solution for that problem. For example, imagine a very smooth and regular landscape with a single hilltop. This is the typical fitness landscape of an easy problem, where all searching strategies are able to find the top of the hill in a straightforward manner. The opposite is true for a very rugged landscape, with many hills which are not as high as the best one. In this case, searching strategies will probably get stuck in local optima.

However, in practical situations, FLs are impossible to visualize, both because of the vast size of the search space and because of the multi-dimensionality of the neighborhood. For this reason, researchers have introduced a set of mathematical measures,

able to capture some characteristics of FLs and express them with single numbers (Vanneschi (2004)). Although none of these measures is capable of expressing completely the vast amount of information that characterizes a FL, some of them revealed to be reliable indicators of the difficulty of problems, for instance: autocorrelation (Weinberger (1990)), entropic measure of ruggedness (EMR) (Vassilev (2000); Vassilev et al. (2000, 2003)), density of states (Rosé et al. (1996)), fitness-distance correlation (Jones and Forrest (1995); Vanneschi (2004)), length of adaptive walks (Stadler (2002)), and basins of attraction size (Ochoa et al. (2010)), plus various measures based on the concepts of fitness clouds (Vanneschi (2004)) and local optima networks (Ochoa et al. (2008)).

Despite having been applied to standard neural networks (Rakitianskaia et al. (2016); Gallagher (2001)), to the best of our knowledge there has been no previous work done applying FLs either to unseen data or neuroevolution.

2.4.1 Autocorrelation

The Autocorrelation coefficient (ρ) (Equation 2.1) is used to measure the ruggedness of the landscape (Weinberger (1990)). It is applied over a time series of fitness values produced by a walk on the landscape with a time-step k which indicates how far apart temporally each solution is. A walk on a FL is a sequence of solutions $(s_0, s_1, ..., s_n)$ such that, for each $t = 1, 2, ..., n$, s_t is a neighbor of s_{t-1} or, in other words, s_t is obtained by applying a mutation to s_{t-1}.

$$\rho(k) = \frac{E[f(s_t)f(s_{t+k})] - E[f(s_t)]E[f(s_{t+k})]}{\sqrt{E[f(s_t)^2] - [E[f(s_t)]]^2} - \sqrt{E[f(s_{t+k})^2] - [E[f(s_{t+k})]]^2}}, \tag{2.1}$$

For walks with a finite length of n, the autocorrelation coefficients can be calculated as follows:

$$\hat{\rho}(k) = \frac{\sum_{t=1}^{n-k}(f(s_t) - \bar{f})(f(s_{t+k}) - \bar{f})}{\sqrt{\sum_{t=1}^{n}(f(s_t) - \bar{f})^2}\sqrt{\sum_{t=1}^{n}(f(s_{t+k}) - \bar{f})^2}}, \tag{2.2}$$

where $\bar{f} = \frac{1}{n}\sum_{t=1}^{n} f(s_t)$, and $n \gg 0$. The longer the walk, the more accurate the estimation becomes.

2.4.2 Fitness clouds

Fitness clouds, introduced by Verel et al. (2003), consist of a scatterplot that maps the fitness of individuals to the fitness of their neighbors that were obtained by the application of a genetic operator. The shape of this scatterplot can give an indication of the evolvability of the genetic operators used and thus some hints about the difficulty of the problem.

Considering a set of individuals $S = \{s_1, s_2, ..., s_n\}$, a set of neighbors of each individuals S_i, $V(s_i) = \{v_1^i, v_2^i, ..., v_{mi}^i\}, \forall i \in [1, n]$ and a fitness function f, the set of points

that form the fitness cloud can be defined as follows:

$$C = \{(f(s_i), f(v_k^i)), \forall i \in [1, n], \forall k \in [1, m_i]\} \tag{2.3}$$

As explained by Vanneschi et al. (2004), fitness clouds also implicitly give information about the genotype to phenotype mapping. The set of genotypes that have the same fitness is called a neutral set (Kimura (1983)), which can be represented by an abscissa in the fitness/fitness plane. According to this abscissa, a vertical slice from the cloud represents the set of fitness values that could be reached from this set of neutrality. For a given offspring fitness value f, a horizontal slice represents all the fitnesses from which one can reach f.

2.4.3 Negative slope coefficient

Negative slope coefficient (NSC) is a measure proposed by Vanneschi (Vanneschi (2004); Vanneschi et al. (2004, 2006)) as an improvement to fitness clouds. Fitness clouds have the downside of only helping determine certain characteristics, but they are incapable of quantifying them. NSC is an algebraic measure that gives an indication of the hardness of the problem, and is defined as follows: The abscissas from the fitness cloud can be partitioned into m segments $\{I_1, I_2, ..., I_m\}$ of equal length. From these segments, one can create the set $\{J_1, J_2, ..., J_m\}$, where each J_i contains all the ordinates corresponding to the abscissas contained in I_i. Let $\{M_1, M_2, ..., M_m\}$ be the averages of the abscissa values contained inside the segments $\{I_1, I_1, ..., I_m\}$ and let $\{N_1, N_2, ..., N_m\}$ be the averages of the ordinate values in $\{J_1, J_2, ..., J_m\}$. We can then define the set of segments $\{S_1, S_1, ..., S_{m-1}\}$, where each S_i connects the point (M_i, N_i) to the point (M_{i+1}, N_{i+1}). For each one of these segments S_i, the slope P_i can be calculated as follows:

$$P_i = \frac{N_{i+1} - N_i}{M_{i+1} - M_i} \tag{2.4}$$

Finally, we can define the NSC as:

$$NSC = \sum_{i=1}^{m-1} min(P_i, 0) \tag{2.5}$$

If $NSC = 0$ then the problem is deemed to be easy; if $NSC < 0$ than the value of NSC quantifies the difficulty of the problem with lower values meaning harder problems.

This original definition proved to be unreliable in some problems, so a new version using size driven bisection was proposed (Vanneschi et al. (2006)). This version differs in the creation of the segments. Instead of partitioning the cloud in m segments, it is split into two segments and the bisection algorithm is applied to both these segments until one of the two following conditions is satisfied: either a segment contains a smaller number

of points than a prefixed threshold, or a segment has become smaller than a prefixed minimum size.

2.4.4 Entropic Measure of Ruggedness

Unlike other statistical measures such as autocorrelation, the entropic measure of ruggedness (EMR), introduced by Vassilev (Vassilev (2000); Vassilev et al. (2000, 2003)), is not, properly speaking, a measure of ruggedness of the landscape, but more of an indicator of the relationship between ruggedness and neutrality.

Assuming that a walk of length n performed on the landscape generates a time series of fitness values $\{f_t\}_{t=0}^{n}$, that time series can be represented as a string $S(\varepsilon) = \{s_1 s_2 ... s_n\}$, where, for each $i = 1, 2, ..., n$, $x_i \in \{\bar{1}, 0, 1\}$. For each $i = 1, 2, ..., n$, $x_i \in S(\varepsilon)$ is obtained using the following function:

$$s_i = \Psi_{f_t}(i, \varepsilon) = \begin{cases} \bar{1}, & f_i - f_{i-1} < -\varepsilon \\ 0, & |f_i - f_{i-1}| \leq -\varepsilon \\ 1, & f_i - f_{i-1} > -\varepsilon \end{cases} \tag{2.6}$$

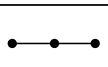
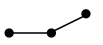
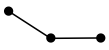
Encoding	00	01	0$\bar{1}$	10	11	1$\bar{1}$	$\bar{1}$0	$\bar{1}$1	$\bar{1}\bar{1}$
Classification	neutral	rugged	rugged	rugged	smooth	rugged	rugged	rugged	smooth
Shape									

Figure 2.3: Classification and encoding of three-point objects

where ε is a real number that determines the accuracy of the calculation of $S(\varepsilon)$, and increasing this value results in increasing the neutrality of the landscape. The smallest possible ε for which the landscape becomes flat is called the information stability and is represented by ε^*.

Using $S(\varepsilon)$, the entropic measure $H(\varepsilon)$ can be calculated as follow:

$$H(\varepsilon) = -\sum_{p \neq q} P_{[pq]} \log_6 P_{[pq]}, \tag{2.7}$$

where p and q are elements from the set $\{\bar{1}, 0, 1\}$, $P_{[pq]}$ is defined as:

$$P_{[pq]} = \frac{n_{[pq]}}{n}, \tag{2.8}$$

where $n_{[pq]}$ is the number of pq sub-blocks in $S(\varepsilon)$ and n is the total number of sub-blocks in $S(\varepsilon)$.

The output from $H(\varepsilon)$ is a value in range $[0, 1]$ that represents an estimate of the variety of shapes in the walk (figure 2.3), with a higher entropy value meaning a larger variety and thus a more rugged landscape.

In this definition, for each walk that is performed, $H(\varepsilon)$ is calculated for multiple ε values, usually $\{0, \frac{\varepsilon^*}{128}, \frac{\varepsilon^*}{64}, \frac{\varepsilon^*}{32}, \frac{\varepsilon^*}{16}, \frac{\varepsilon^*}{8}, \frac{\varepsilon^*}{4}, \frac{\varepsilon^*}{2}, \varepsilon^*\}$, and then the mean of $H(\varepsilon)$, represented as $\bar{H}(\varepsilon)$, over all performed walks is calculated for each value of ε. A new proposed version of this method (Malan and Engelbrecht (2009)) aimed at reducing the characterization of the landscape to a single scalar. To characterise the ruggedness of a function, the following single value measure is proposed for a fitness function f:

$$R_f = \max_{\forall \varepsilon \in [0, \varepsilon^*]} H(\varepsilon) \tag{2.9}$$

To approximate the theoretical value of R_f , the maximum of $\bar{H}(\varepsilon)$ is calculated for all ε values.

2.4.5 Gradient Measures

Recently, two gradient analysis methods were proposed (Rakitianskaia et al. (2016); Malan and Engelbrecht (2013)). While ruggedness quantifies the presence of fitness changes in the landscape, it does not quantify the magnitude of the jumps, something these measures aim to do.

The first measure, average estimated gradient G_{avg}, measures the mean magnitude of change in fitness values, and is calculated as follows:

$$G_{avg} = \frac{\sum_{t=0}^{T-s} |g(t)|}{T} \tag{2.10}$$

where T is the number of steps in the walk, s is the step size, and $g(t)$ can be defined as:

$$g(t) = \frac{\Delta e_t}{s} \tag{2.11}$$

where Δe_t is the difference between the error values of the weight vectors defining step t of the random walk.

The second measure, standard deviation of gradient G_{dev}, measures the standard deviation associated to G_{avg}, and is defined as:

$$G_{dev} = \sqrt{\frac{\sum_{t=0}^{T-s} |G_{avg} - |g(t)||^2}{T - s}} \tag{2.12}$$

Lower values for both these measures usually indicate a simpler landscape that is easier to search. However, a lack of gradient can also mean high neutrality in the landscape.

Chapter 3

Methodology

This chapter describes the methodology that was used in this work. It covers the creation of our proposed neuroevolution algorithm, the way we perform walks and sample points, new fitness landscape analysis measures, how we defined the experiments and what problems we used.

3.1 Defining the Neuroevolution Algorithm

We decided to use a grammar-based approach when developing the algorithm because it provides the modularity and flexibility needed for our experiments (*e.g.* it's possible to generate both an Auto-Encoder and a CNN with the same grammar as both use the same type of layers). In grammatical encoding strategies, information regarding the types and parameters of the layers/network is encoded in a grammar, meaning that if we desire to change parameter values of the network, we only need to change the grammar. The grammar used for this work is shown in Figure 3.1.

Another advantage of using this grammatical approach is related to the discretization of the parameter values. Discretizing the range and possibilities of values that each parameter can have greatly reduces the search space while keeping the quality of the solutions under control, as in most cases, intermediate values can have little to no significant effect in the effectiveness of the solutions (Gómez et al. (2018)).

Genotype encoding

We view the genotypes as being composed of two different subsections, S_1 and S_2, that are connected using the Flatten gene. The Flatten gene implements the conversion (i.e., the "flattening") of multidimensional feature matrices from the Convolutional layers into a one-dimensional array that is fed to the following Dense layer. On S_1 we have genes that encode the layers that deal with feature extraction from the images, Convolutional and Pooling layers, and on S_2 we have genes that encode classification and pruning, Dense and Dropout layers.

Conv ::	filters	|32,64,128,256|
	kernel_size	|2,3,4,5|
	stride	|1,2,3|
	activation	|relu, elu, sigmoid|
	use_bias	|true, false|
Pool ::	type	|Max, Avg|
	pool_size	|2,3,4,5|
	stride	|1,2,3|
Dense ::	units	|8,16,32,64,128,256,512|
	activation	|relu, elu, sigmoid|
	use_bias	|true, false|
Dropout ::	rate	$[0.0 \rightarrow 0.7]$
Optimizer ::	learning_rate	|0.01, 0.001, 0.0001, 0.00001|
	decay	|0.01, 0.001, 0.0001, 0.00001|
	momentum	|0.99, 0.9, 0.5, 0.1|
	nesterov	|true, false|

Figure 3.1: Grammar used to evolve the CNNs presented in this work

This separation allows for a more balanced generation of the genomes and easier application of the genetic operators. Apart from the Flatten layer, the only other layer with a semi-fixed model is the last layer, that will always be a Dense with softmax activation and a number of units equal to the number of classes to be predicted, having only the bias available for mutation.

Besides the Flatten layer, the only other layer that is the same for all possible individuals is the output layer, which is a Dense layer with softmax activation and a number of units equal to the number of classes to be predicted. The genetic operators cannot modify this layer, except for the bias parameter.

Before evaluation, a genotype is mapped into a phenotype, that is a neural network itself, and all weights are initialized following the Glorot initialization method (Glorot and Bengio (2010)). An example genotype can be seen in Figure 3.2.

Genetic Operators

Due to the difficulty of defining a crossover-based neighborhood for studying FLs (Gustafson and Vanneschi (2005)), we consider only mutation operators. Given the vast amount of mutation choices in neuroevolution, we restrict our study to three different types of operators:

- **Topological mutations:** Mutations that add or delete a gene, except for the Flatten gene, changing the topology of the network.
- **Parameter mutations:** Mutations that change the parameters encoded in a gene. They cover all parameters of all gene types, excluding the Flatten gene (which has no parameters).

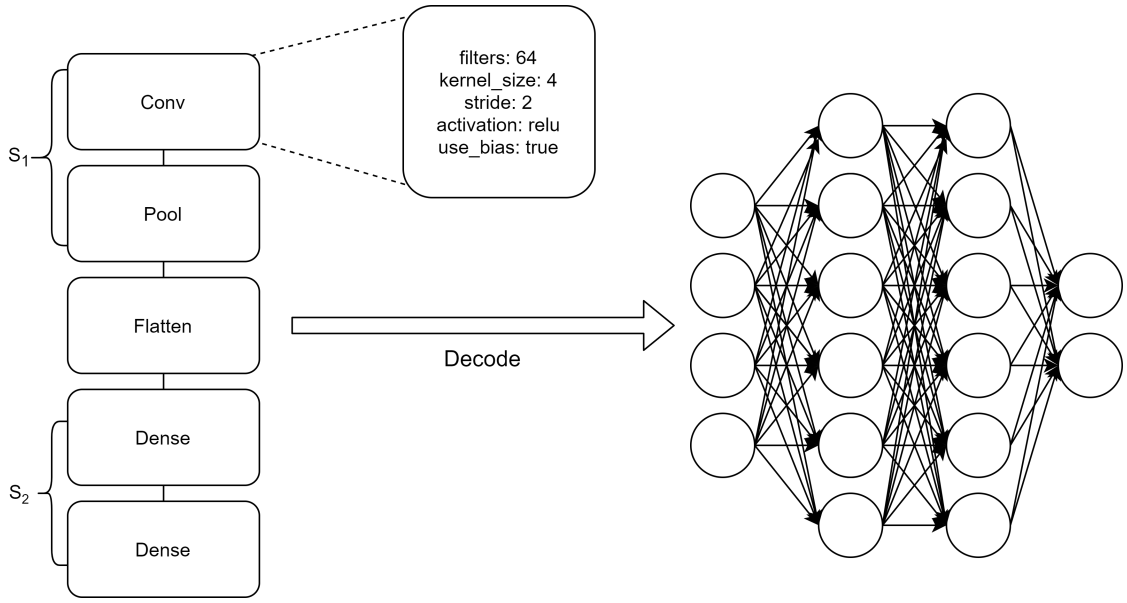


Figure 3.2: Example of a genome built by the grammar and the network it produces when decoded

- **Learning mutations:** Mutations that change the Optimizer's parameters that guide the learning of the networks. These parameters are encoded in the Optimizer gene.

Evaluation

Since we are working with multiclass classification problems (Described in section 3.3.1) that are not one-hot encoded, we use Sparse Categorical Cross-Entropy as a loss function, which motivates the need to have the fixed number of neurons and activation function in the output layer. During the evolutionary process, we use the loss value on the training set as the fitness function to evaluate the networks. We also measure the accuracy and loss in a separate test set in order to study the generalization ability.

Regarding the chosen Optimizer, we decided to use Stochastic Gradient Descent (SGD) due to two important factors: the range of values for the SGD parameters has been extensively tested so the value choice could be done more accurately, and, although ADAM (Kingma and Ba (2015)) is more common nowadays due to achieving better overall better results, SGD outperforms ADAM when it comes to generalization capability (Wilson et al. (2017)) and as we previously mentioned it was in our interest to study the generalization capability.

3.2 Defining the Landscape analysis measures

Given the huge size of the neuroevolution search space, and in the attempt to generate walks that are, as much as possible, representatives of the portions of the search space

explored by the evolutionary algorithm, in this work we have decided to calculate autocorrelation using selective walks (Vanneschi (2004)). Unlike random walks, selective walks apply some selection pressure, but still less than adaptive walks. In selective walks, for each $t = 1, 2, ..., n$, s_t is a selected solution from the neighborhood of s_{t-1}. To apply selection pressure to the neighbors, tournament selection is used; in other words, s_t is the best solution (i.e., the one with the best fitness on the training set) in a sample of m randomly generated neighbors of s_{t-1}.

3.2.1 Sampling Methodology

Many of the previously discussed measures can be calculated using a sample of the search space. However, as discussed in Vanneschi (2004), the sampling methodology used to generate this set of individuals may be crucial in determining the usefulness of the measures. As in Vanneschi (2004), we also use a Metropolis-Hastings (Madras (2002)) approach in this work. Similarly to selective walks, this methodology has a selection pressure. This characteristic makes Metropolis-Hastings sampling preferable compared to a simple uniform random sampling. Particularly in our study, given the huge size of the neuroevolution search space, uniform random sampling is unlikely to generate a sample that may represent the search space portion explored by the EA. A good example is a plateau: with uniform random sampling, multiple points with the same fitness could be re-sampled, while using Metropolis-Hastings, only one point would be sampled and then the algorithm would look for better solutions. Since neuroevolution works with unbounded loss values, we have performed some changes on the Metropolis-Hastings algorithm, compared to the version used in Vanneschi (2004). The method we use is described in Algorithm 1.

Algorithm 1: Metropolis-Hastings sampling.

```
Sample size = n
Generate a random individual x_1
for t = 1,2,...,n do
    Generate a random candidate individual x'
    Calculate the acceptance ratio α = norm(x') / norm(x_t)
    if α ≥ u ∈ σ(0,1) then
        Accept the candidate, x_{t+1} = x'
    else
        Reject the candidate, x_{t+1} = x_t
```

We normalize the fitness values using the following expression: $norm(x) = \frac{1}{1+f(x)}$, where $f(x)$ is the fitness of the individual x. This normalization is done since we are working with loss values in the range $[0, \infty]$.

3.2.2 Density Clouds

We propose the use of a new measure called density clouds as an alternative to the density of states. Density clouds allow us to use the same sample that is used for fitness clouds, whereas density of states would require a much larger sample since we would need to find neighbours in that sample, a very hard task in such a large space of possibilities. Density clouds use the same points as the fitness clouds to produce a visual representation of the shape and density of that distribution. The visual interpretation of density clouds is essentially the same as for fitness clouds, but concentrated on the areas with the largest density of points, with the added advantage that the plots will still be easily interpretable in a space of thousands of samples.

Let $\{X_1, X_2, ..., X_n\}$ be a univariate independent and identically distributed sample drawn from some distribution with an unknown density f. The kernel density estimator is:

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h}), \tag{3.1}$$

where K is a kernel function and h is the bandwidth.

3.2.3 Overfitting Measure

Vanneschi et al. (2010) proposed a new measure to quantify overfitting during the evolution of genetic programming algorithms. We propose using this measure with slight modifications and apply it to selective walks to predict the generalization ability of the models. Algorithm 2 describes the measure, that will be called OverFitting Measure (OFM) from now on. Intuitively, the amount of overfitting measured at a given moment is based on the difference between training and test fitness, and how much this difference has increased since the best test point (btp), i.e., the moment in which the test fitness was at its best (lines 12–14). The btp is set at the beginning of the walk, with overfitting being 0 at that moment (lines 2–3). Every time a new best test fitness is found, the btp is updated to that point of the walk, and overfitting is again set to 0 (lines 8–10). Overfitting is also set to 0 when the training fitness is worse than the test fitness (lines 5–6). Because the selective walk does not use elitism, we are bounding the amount of overfitting to 0, otherwise, it would be possible to arrive at negative values (*max* at line 14).

Algorithm 2: Method used to measure overfitting in minimization problems.

```
Length of the walk = n
Best test point, btp = 0
overfit(btp) = 0
for i = 1,2,...,n do
    if training_fit(i) > test_fit(i) then
        overfit(i) = 0
    else
        if test_fit(i) < test_fit(btp) then
            btp = i
            overfit(btp) = 0
        else
            diff_now = |training_fit(i) − test_fit(i)|
            diff_btp = |training_fit(btp) − test_fit(btp)|
            overfit(i) = max(0, diff_now − diff_btp)
```

3.3 Experimental Study

3.3.1 Dataset selection and creation

Table 3.1 describes the main characteristics of the datasets used as test cases in our experiments. The partition into training and test set is made randomly, and it is different at each run. For all datasets, a simple image scale adjustment was done, setting pixel values in the $[0, 1]$ range. No further data pre-processing or image augmentation was applied to the datasets. The MNIST dataset consists of a set of grayscale images of handwritten digits from 0 to 9 (LeCun and Cortes (2010)). Fashion-MNIST (FMNIST) is similar to MNIST, but instead of having digits from 0 to 9, it contains images of 10 different types of clothing articles (Xiao et al. (2017)). CIFAR10 contains RGB pictures of 10 different types of real-world objects (Krizhevsky (2012)). SVHN contains RGB pictures of house numbers, containing digits from 0 to 9 (Netzer et al. (2011)). SM (small and mislabelled) is a hand-tailored dataset that we have artificially created to have a case in which neuroevolution overfits. It was created by taking the last 30% of samples from MNIST and applying label corruption. Half of the values from each odd label were changed to another label value. Label 1 was changed into 3, 3 became 9, 5 became 0, 7 became 4, and 9 became 1. The new labels were associated randomly to avoid any human bias.

3.3.2 Experimental settings

From now on, we will use the term *configuration* to indicate an experiment in which a particular type of mutation was used on a particular dataset. For each configuration, we

Table 3.1: Number of training and test observations, and number of classes of each dataset.

	Training set	**Test set**	**Classes**
MNIST	60000	10000	10
FMNIST	60000	10000	10
CIFAR10	50000	10000	10
SVHN	73257	26032	10
SM	18000	3000	10

Table 3.2: Parameter values used in our experiments.

Selective walk		**neuroevolution**		**Learning**	
Length	30	Population size	10	Epochs	8
# Neighbors	3	# Generations	20	Batch	64
		Tournament size	2		
Sampling		Mutation rate	1		
Size	500	Crossover rate	0		
# Neighbors	5	No elitism			

generated samples of 500 individuals, 10 independent selective walks and 10 independent neuroevolution runs. All neuroevolution runs are performed starting with a randomly initialized population of individuals, and all the selective walks are constructed starting with a randomly generated individual.

To determine the values of the main parameters (e.g., population size and the number of generations for neuroevolution, length of the walk and number of neighbors for selective walks, etc.) we have performed a preliminary experimental study with multiple values and selected the ones that allowed us to obtain results in "reasonable" time[1] with our available computational resources[2]. However, we do acknowledge that as with any evolutionary algorithm, higher values for these parameters could produce stronger and more accurate results.

The employed parameter values are reported in Table 3.2. The first column contains the parameters used to perform the selective walks and sampling, while the second column contains the parameters of the neuroevolution. One should keep in mind that, in order to evaluate all the neural networks in the population, all the networks need to go through a learning phase at each generation of the evolutionary process. The third column reports the values used by each one of those networks for learning.

[1] On average, 5 hours per run and 26 hours per sampling.

[2] Our experiments were performed on a machine with a gtx 970, a gtx 2070 and 16GB of RAM.

Chapter 4

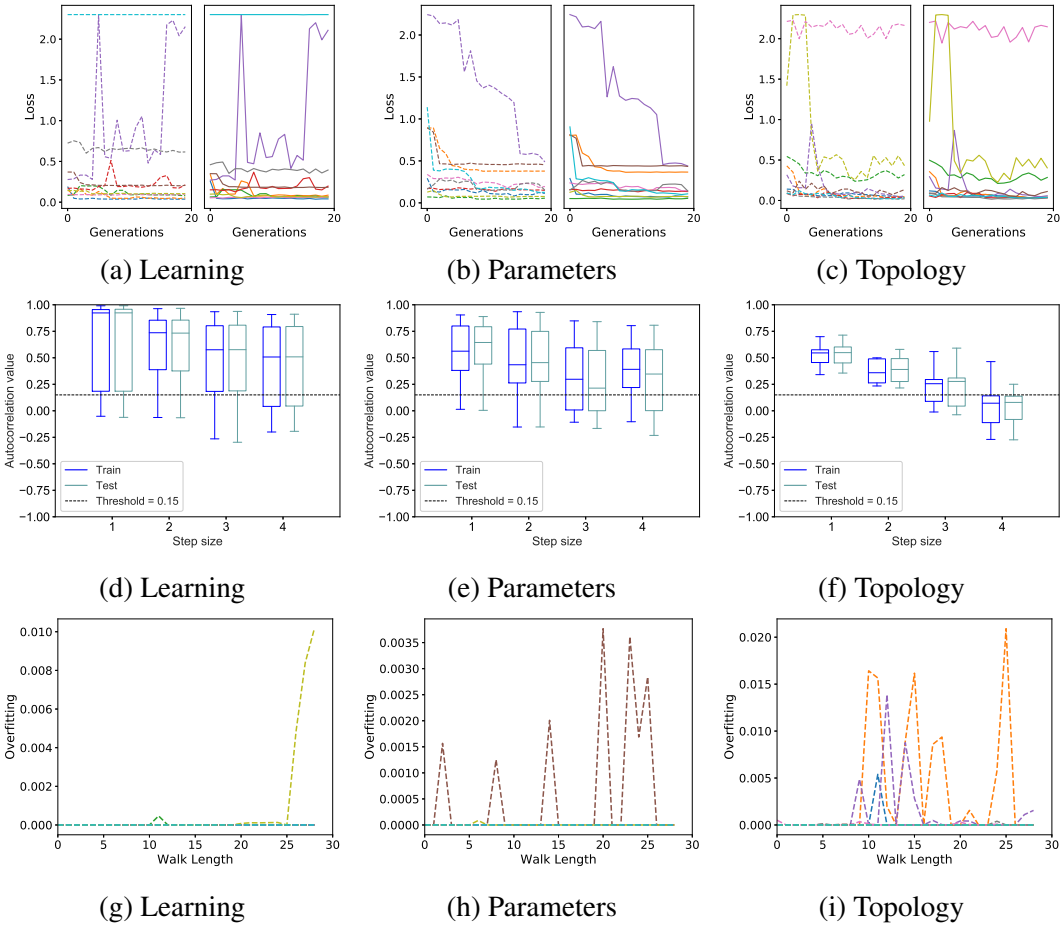
Experimental Results

We now describe the results obtained in our experiments. We analyse the predictive value of the different measures we use to characterize fitness landscapes, observing the results obtained in each problem, and then we briefly comment on the general performance of the different types of mutation used.

Autocorrelation and Overfitting Measure

We begin by analyzing the ability of autocorrelation to characterize training and test performance of neuroevolution of CNNs, as well as the ability of OFM to detect and measure overfitting. Fig. 4.1 reports the evolution of the loss, the autocorrelation and the OFM for the MNIST problem. The first line of plots reports the evolution of the loss against generations for the three studied mutation operators. Each plot in the first line is partitioned into two halves: the leftmost one reports the evolution of the training loss of the best individual, while the rightmost one reports the loss of the same individual, evaluated on the unseen test set. Each curve in these plots reports the results of one neuroevolution run. The second line contains the boxplots of the autocorrelation values, calculated over 10 independent selective walks, both on the training and on the test set. The third line reports the evolution of the OFM value against the length of a selective walk for each different mutation type. Each curve reports the results of a single walk. Each column of plots reports the results for a different type of mutation, allowing us to easily compare the outcome of the neuroevolution and the one of the studied measures for the different configurations.

As we can observe from plots (a) and (b) of Fig. 4.1, when we employ learning mutation and parameters mutation, the MNIST problem is easy to solve, both on training and test set. This is confirmed by the fact that fourteen out of twenty runs have a loss value below 0.2 which translates in accuracy values ranging from 94% up to 98%. Furthermore, the evolution of the loss suggests that given more generations these values would improve since the majority of the runs was still, slowly, converging towards zero. Now, looking at plots (d) and (e), we observe that the autocorrelation captures the fact that the problem is



(a) Learning (b) Parameters (c) Topology

(d) Learning (e) Parameters (f) Topology

(g) Learning (h) Parameters (i) Topology

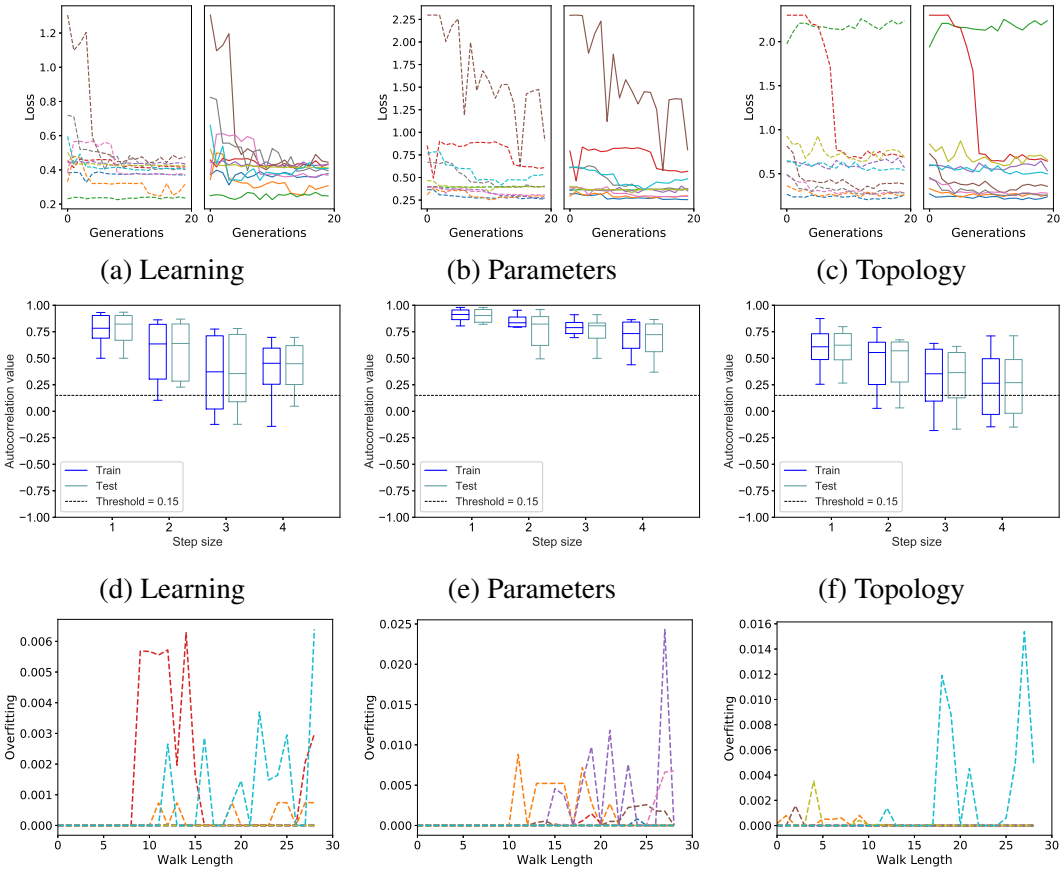
Figure 4.1: *MNIST dataset*. Plots (a), (b) and (c): Neuroevolution results; plots (d), (e) and (f): autocorrelation results; plots (g), (h) and (i): results of the measure of overfitting (OFM). Plots (a), (d) and (g): results for the learning mutation; plots (b), (e) and (h) for the parameters mutation; plots (c), (f) and (i) for the topology mutation. Remarks: plots (a), (b) and (c) report the evolution of the best fitness in the population at each generation (one curve for each performed neuroevolution run). Each plot is partitioned into two subfigures: training loss on the left and test loss on the right. Plots (d), (e) and (f) report the boxplots of the autocorrelation, calculated over 10 independent selective walks, along with the threshold line at 0.15. Plots (g), (h) and (i) report the evolution of the OFM value for each of the 10 independent selective walks.

easy. In fact, in both cases, practically the whole autocorrelation box stands above (and the medians never go below) the 0.15 threshold. When the topology mutation is used, the situation changes: the number of runs in which the evolution does not have a regular trend is larger. This may not be obvious by looking at plot (c), because of the scale of the y-axis, but the lines are now much more rugged than they were for the other two cases. The problem is now harder than it was, and as we can see in plot (f), the autocorrelation catches this difficulty. In particular, we can observe that when the step is equal to 4, the whole autocorrelation boxes are below the threshold. Finally, looking at plots (g) to (i),

we observe that OFM captures the fact that neuroevolution does not overfit for the MNIST dataset. In fact, the OFM values are always low, and keep returning to 0.

The partial conclusion that we can draw for the MNIST dataset is that learning and parameter mutations are more effective operators than topology mutations, and this is correctly predicted by the autocorrelation. Furthermore, we can observe that the neuroevolution results obtained on the test set are very similar to the ones on the training set, practically for all the runs we have performed. Also, this feature is captured by the autocorrelation, since the training and test boxes are very similar to each other for practically all the configurations. This is an indication of lack of overfitting, and this feature is correctly measured by the OFM.

Now we consider the results obtained for the FMNIST dataset, reported in Fig. 4.2.



(a) Learning (b) Parameters (c) Topology

(d) Learning (e) Parameters (f) Topology

(g) Learning (h) Parameters (i) Topology

Figure 4.2: *FMNIST dataset*. The organization of the plots is analogous to Figure 4.1.

Describing the results for this dataset is straightforward: observing the neuroevolution plots, we can see that for the three configurations the problem is easy, both on training and test set. In fact, all the curves are steadily decreasing and/or close to zero. For the learning operator, all runs ended with loss values below 0.5, which translates in accuracy values ranging from 85% up to 92%. For both parameter and topology operators, a total of twelve

out of twenty runs report loss values under 0.5. Observing the scale on the left part of the plots, we can also observe that when topology mutation is used (plot (c)), the problem is slightly harder than when the other mutations are used, since the achieved values of the loss are generally higher. All this is correctly measured by the autocorrelation, given that the boxes are above the threshold for all the configurations, and, in the case of the topology mutation (plot (f)), they are slightly lower than in the other cases. Last but not least, also in this case, training and test evolution of loss are very similar between each other, and this fact finds a precise correspondence in the autocorrelation results, given that the training boxes are generally very similar to the test boxes. This also indicates no overfitting, and this fact is correctly captured by the OFM, that shows values that keep returning to 0. All in all, we can conclude that also for the FMNIST dataset, autocorrelation is a reliable indicator of problem hardness and the OFM correctly predicts lack of overfitting.

The results for the CIFAR10 dataset are reported in Fig. 4.3.

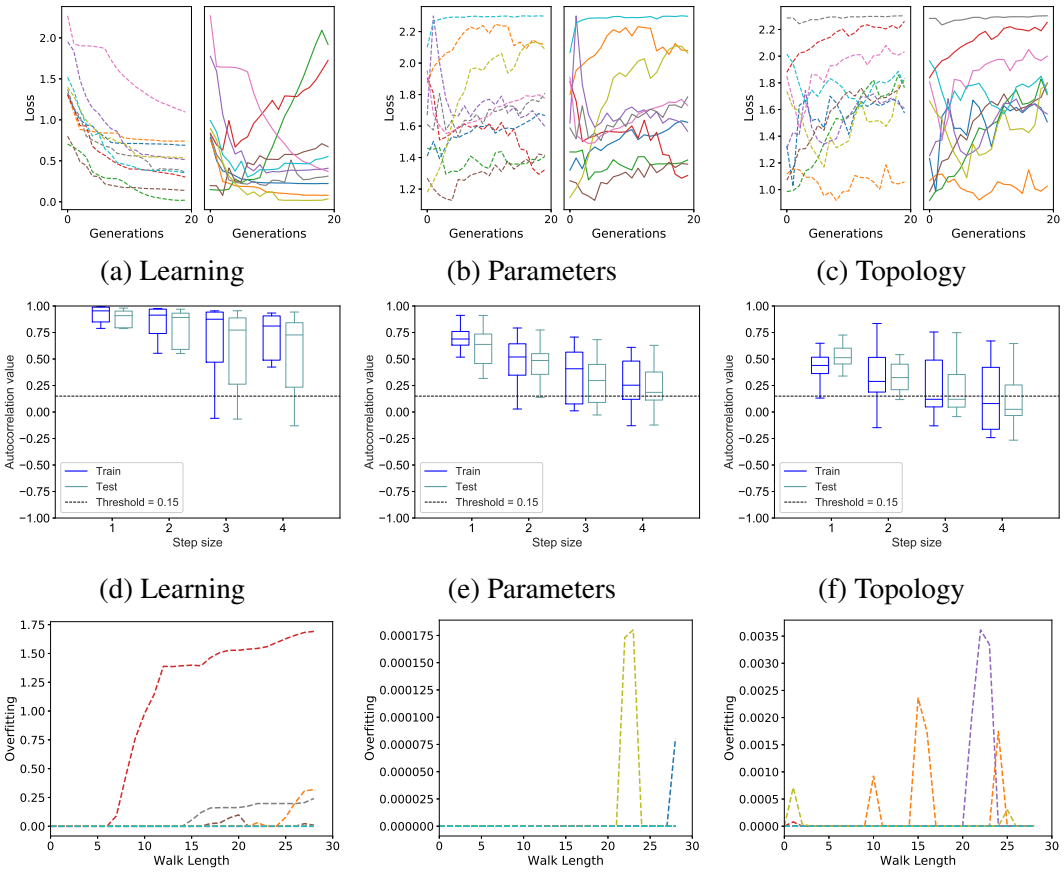
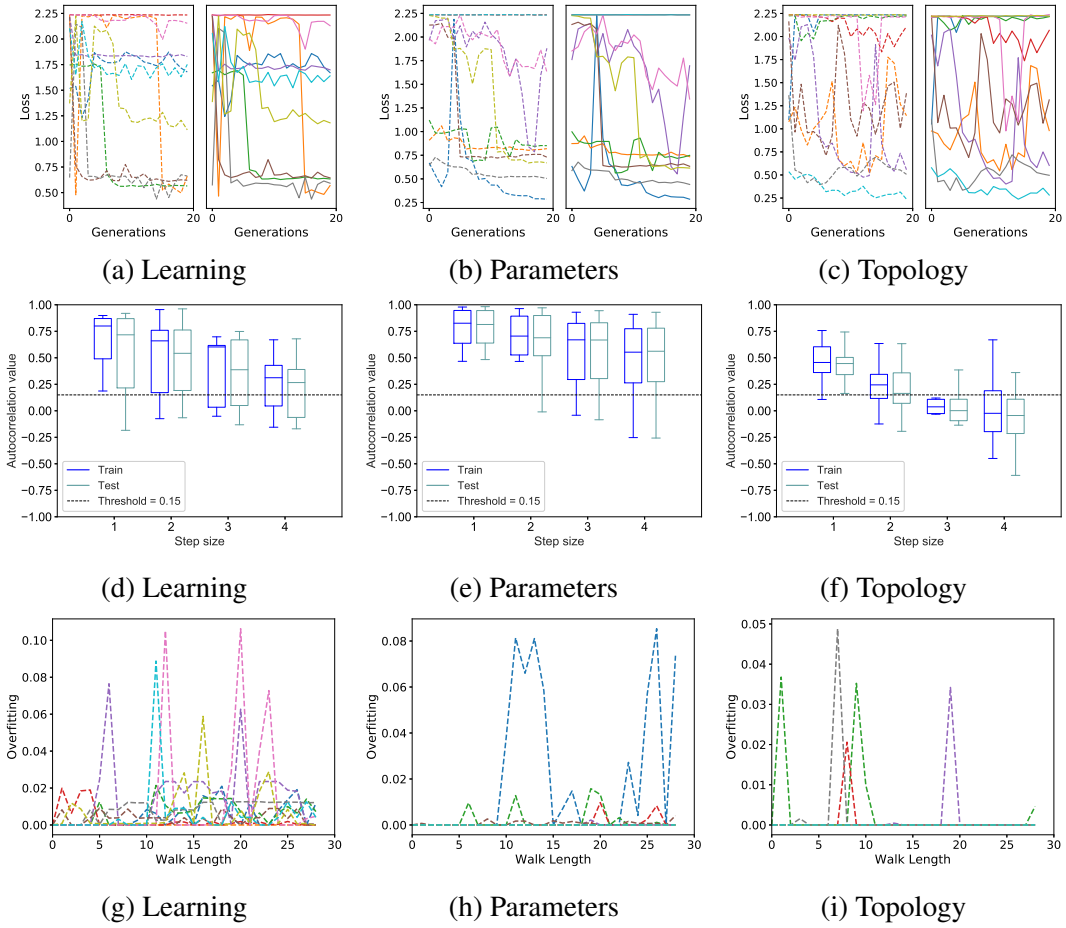


Figure 4.3: *CIFAR10 dataset.* The organization of the plots is analogous to Figure 4.1.

Observing the neuroevolution results, we can say that when the learning mutation is used, the problem is substantially easy (almost all the loss curves have a smooth decreasing trend); however, at the same time, among the three types of mutation, learning

mutation is the one in which there is a more marked difference between training and test evolution, which indicates the possibility of overfitting, at least in some runs. Interestingly, on the learning operator, in the training plot only half of the runs present loss below 0.5, but in the test plot, despite the clear overfitting, we have six runs below that threshold, one more than in training. At the same time, when the parameters mutation is used, the problem is uncertain (in some runs the loss curves have a decreasing trend, while in others they have an increasing trend), but the training and test evolution are rather similar between each other in every run. For this operator, loss values are in the range of [1, 2.3], which translates into accuracy values in the range of [62%, 10%]. Finally, when the topology mutation is used, the problem is hard (almost all the loss curves have an increasing trend), but once again no substantial difference between training and test behaviors can be observed. This operator produced loss values highly similar to the ones produced by the parameters mutation, with only a slight change in the lower bound. Loss values are in the range of [1.3, 2.3], which translates into accuracy values in the range of [63%, 10%]. Looking at the autocorrelation results, we find a reasonable correspondence: for the learning mutation all the boxes are clearly above the threshold, for the parameters mutation the boxes are not as high as for the learning mutation, beginning to cross the threshold with steps 3 and 4, and finally, for the topology mutation the boxes are even lower, with the medians below the threshold for steps 3 and 4, and more than half the height of the boxes also below the threshold for step 4. As already observed in plot (f) of Fig. 4.2, longer steps seem to be better indicators, when the autocorrelation is applied to hard problems. The different behavior between training and test set also finds a correspondence in the autocorrelation results (plot (d)), given that the test boxes are taller than the training boxes, in particular for step 4. At the same time, the potential presence of overfitting is clearly detected by the OFM (plot (g)), that assumes growing values in some cases. As for plots (h) and (i), they reflect the absence of overfitting for parameters and topology mutations, as expected. All in all, also for the CIFAR10 dataset the autocorrelation is a reasonable indicator of problem difficulty, while the OFM reveals to be a good measure of overfitting.

The results for the SVHN dataset are reported in Figure 4.4. In this case, the plots of the loss evolution indicate that the problem is uncertain when learning mutation is used (given that approximately half of the curves have a decreasing trend, while the other half have an oscillating trend), easy when parameters mutation is used (with the majority of the curves having a decreasing trend) and hard when topology mutation is used (with most curves exhibiting an oscillatory behaviour, which indicates poor optimization ability). On the learning operator, we have loss values in the range of [0.56, 2.23], which translates into accuracy values in the range of [84%, 19%], and only four runs ended with a loss value below 1.0. Both parameter and topology operators have very close boundary values, both having all runs in the range of [0.23, 2.23] loss wise and [93%, 19%] accuracy wise, with



(a) Learning (b) Parameters (c) Topology

(d) Learning (e) Parameters (f) Topology

(g) Learning (h) Parameters (i) Topology

Figure 4.4: *SVHN dataset*. The organization of the plots is analogous to Figure 4.1.

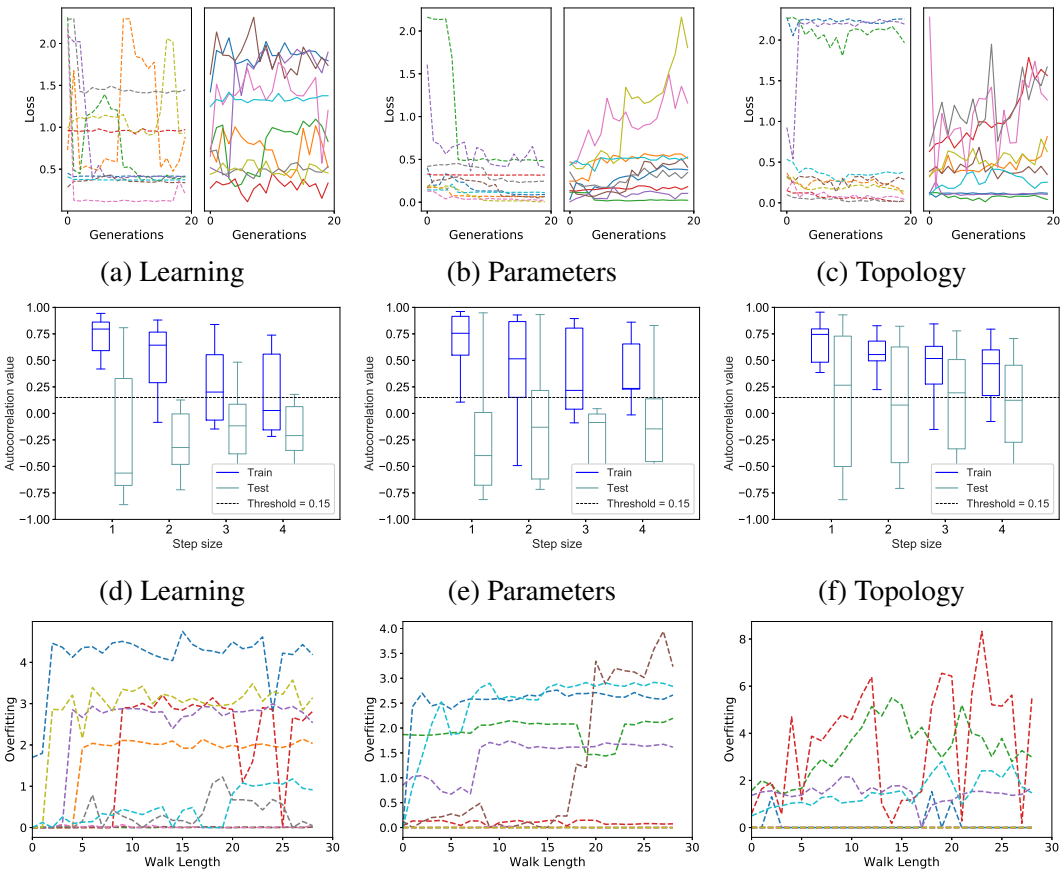
only three runs below 1.0 loss each.

At the same time, some differences, although minimal, can be observed between training and test evolution. Specifically, when a run has an oscillatory behavior, the oscillations tend to be larger on the test set than on the training set (for instance, but not only, in the dark blue curves on plot (a), the violet curves in plot (b) and the red curves in plot (c)). Also, in this case, autocorrelation is confirmed as a reasonable indicator of problem difficulty. In fact, for learning mutation, the boxes are crossing the threshold for steps 3 and 4, for parameters mutation they are above the threshold, and for topology mutation they are almost completely below the threshold for steps 3 and 4. The medians are lower and the dispersion of values is larger for step 4, which reflects well the neuroevolution behavior observed in plot (c) (unstable and often returning to high values of the loss). The highest step size is once again the most reliable. Regarding OFM, it reveals several peaks but no clear trend, either because there is no overfitting or because both training and test fitness values oscillate too much to reveal a trend.

There was also an interesting finding regarding the topology of the best network obtained by the topology operator. Earlier in section 1 we claimed that FL analysis of

neuroevolution could help find configurations better and faster than manually tailored ones. The previously mentioned solution serves as an example of such, where a network composed only by six different size convolutional layers and two small dense layers can outperform solutions with more common topologies that include pooling and dropout layers.

Finally, we analyse the results obtained on the SM dataset, reported in Figure 4.5.



(a) Learning (b) Parameters (c) Topology

(d) Learning (e) Parameters (f) Topology

(g) Learning (h) Parameters (i) Topology

Figure 4.5: *SM dataset*. The organization of the plots is analogous to Figure 4.1.

Looking at plots (a), (b) and (c), we can observe the following facts: first of all, neuroevolution overfits in all the three cases. This was expected, since the SM dataset was explicitly created to have a test case with substantial overfitting, and can be observed looking at the large differences between training and test evolution. In particular, we can see that in some cases the loss curves are steadily decreasing on the training set, while they are either increasing or oscillating on the test set. Secondly, we observe that, for the SM dataset, evolution is harder when using learning mutation, compared to either parameters or topology mutations. Both these observations find a correspondence in the autocorrelation boxplots. In fact, the training and test boxes are visibly different, with the test boxes always positioned lower, and often taller, compared to the training ones. Even more im-

portantly, training boxes are often (completely, or almost completely) above the threshold, while test boxes are always (completely or in large part) below the threshold. At the same time, as we can see in plot (d), for learning mutation and step size equal to 4, the median is lower than the threshold on the training set, and this is the only case in which this event verifies, indicating a bigger difficulty for the learning mutation. Concerning the OFM, it correctly detects the overfitting, producing very high results in all three plots. This again serves as a confirmation that the OFM is able to detect and quantify overfitting. It is also interesting to compare the OFM values for CIFAR10 in the only case where an overfitting trend was observed (Figure 4.3, plot (g)) and in the SM dataset: OFM values are clearly larger for SM, which correctly indicates the larger amount of overfitting observed.

Gradient analysis

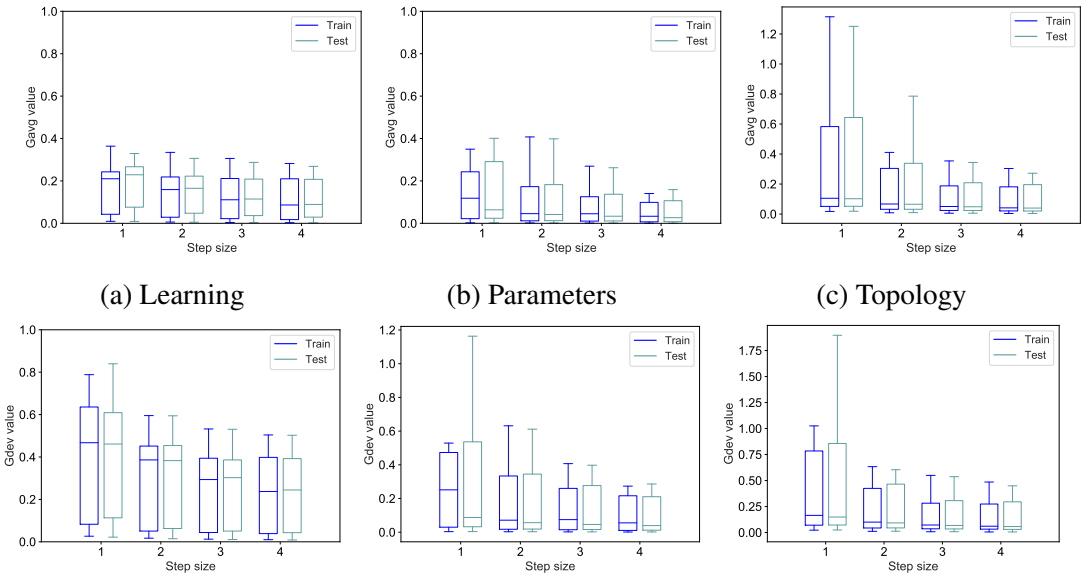
We now analyze the ability of both gradient measures to quantify the ruggedness of the landscape.

For all problems (Figures 4.6 to 4.10) the G_{avg} values were all fairly low regarding the training data, with larger jumps associated to the topology mutation operator, meaning that the landscape is more rugged, in accordance to the results obtained by autocorrelation. Regarding the test values, we have several different and interesting results. For MNIST (Figure 4.6) the values are very similar to the training ones, being only a little bit higher. This can be confirmed by the evolution plots (Figures 4.1d, 4.1e and 4.1f) where both training and test results are very similar, but in some cases, the changes are more abrupt in the test set.

For FMNIST (Figure 4.7) the results are almost the same as MNIST, test values very similar to training values, being only slightly higher due to the more abrupt changes. CIFAR10 (Figure 4.8) shows some very interesting results regarding the learning operator. In Figures 4.3a and 4.3g we can see that there is overfitting, but the autocorrelation was not able to detect it, showing results that indicate a very smooth surface. However, in Figure 4.8a we can clearly see that the jumps in the gradient are very different between training and test data. This result, along with the ones from the SM dataset (Figure 4.10), shows that this measure is capable of providing indications on overfitting.

The results of SVHN (Figure 4.9) were also very interesting. Despite both evolution and autocorrelation results (Figures 4.4a to 4.4f) showing that the landscape from the test set is highly rugged, the jumps in the gradient are very similar to the ones on the training set. This is a clear indication that the models fail to generalize on this problem.

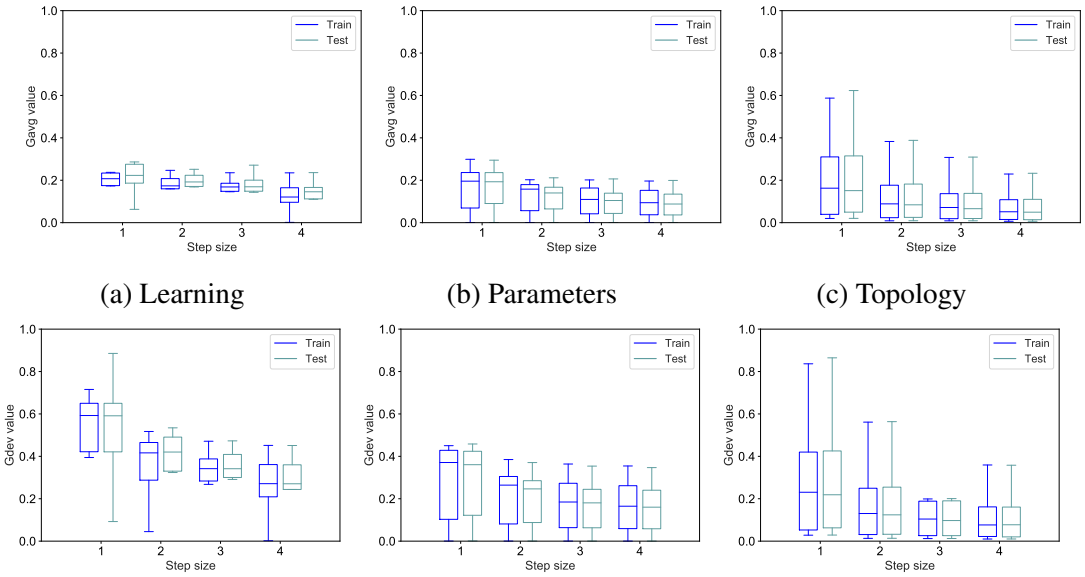
Regarding G_{dev}, if the values are similar to G_{avg} then the landscape should contain sudden jumps (step-like landscape), and if G_{dev} is higher then we are in the presence of accentuated cliffs and/or ravines. For MNIST, despite the smooth landscapes produced by the learning mutation, this measure detects the presence of ravines/cliffs, as the mean G_{dev} is more than double the mean G_{avg}. For FMNIST, we have results very similar to MNIST,



(a) Learning (b) Parameters (c) Topology

(d) Learning (e) Parameters (f) Topology

Figure 4.6: *MNIST dataset*. Plots (a), (b) and (c): G_{avg} results; plots (d), (e) and (f): G_{dev} results. Remarks: Plots (a) to (f) report the boxplots of the gradient measures, calculated over 10 independent selective walks.

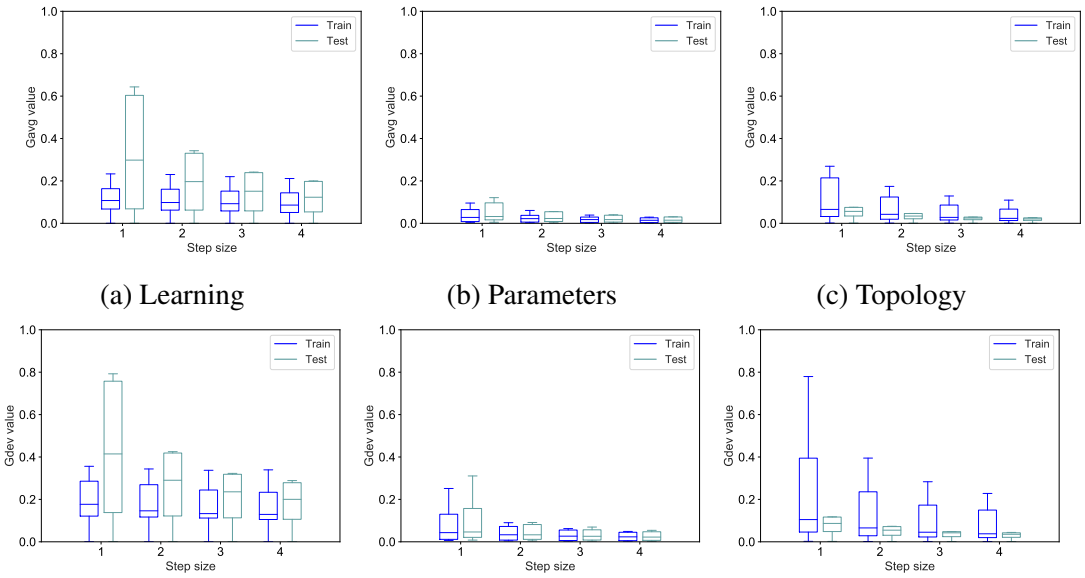


(a) Learning (b) Parameters (c) Topology

(d) Learning (e) Parameters (f) Topology

Figure 4.7: *FMNIST dataset*. The organization of the plots is analogous to Figure 4.6.

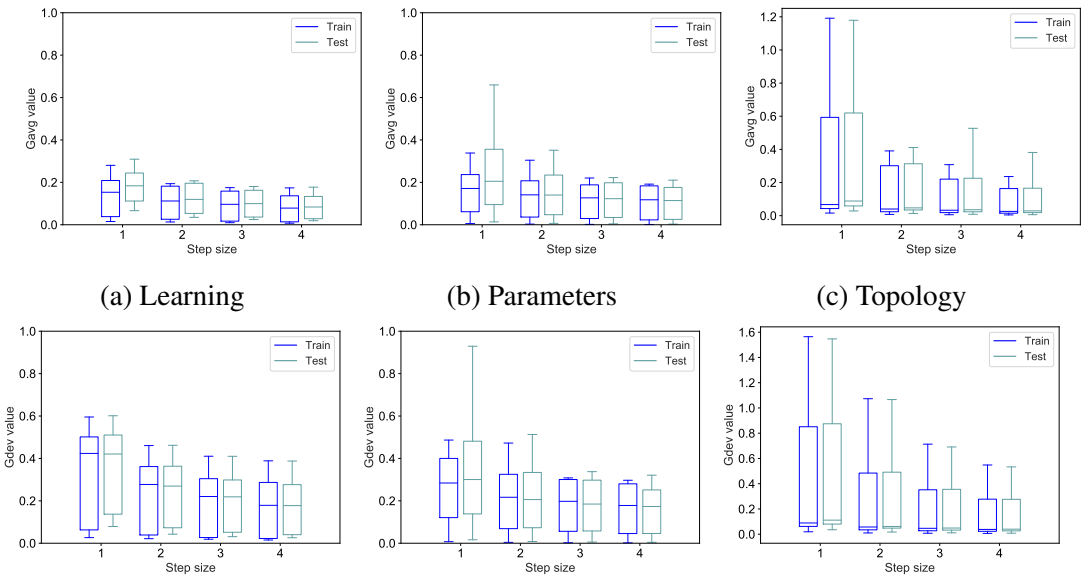
with G_{dev} accusing the presence of cliffs/ravines on the learning mutation. For lower step values on parameter mutation, the discrepancy of values between G_{avg} and G_{dev} is also high but becomes small as we go into larger step sizes. For CIFAR10, despite the differences regarding the upper quartile, the means have similar values to their corresponding G_{avg}, meaning that despite being rugged (the landscapes produced by the parameter and



(a) Learning (b) Parameters (c) Topology

(d) Learning (e) Parameters (f) Topology

Figure 4.8: *CIFAR10 dataset*. The organization of the plots is analogous to Figure 4.6.

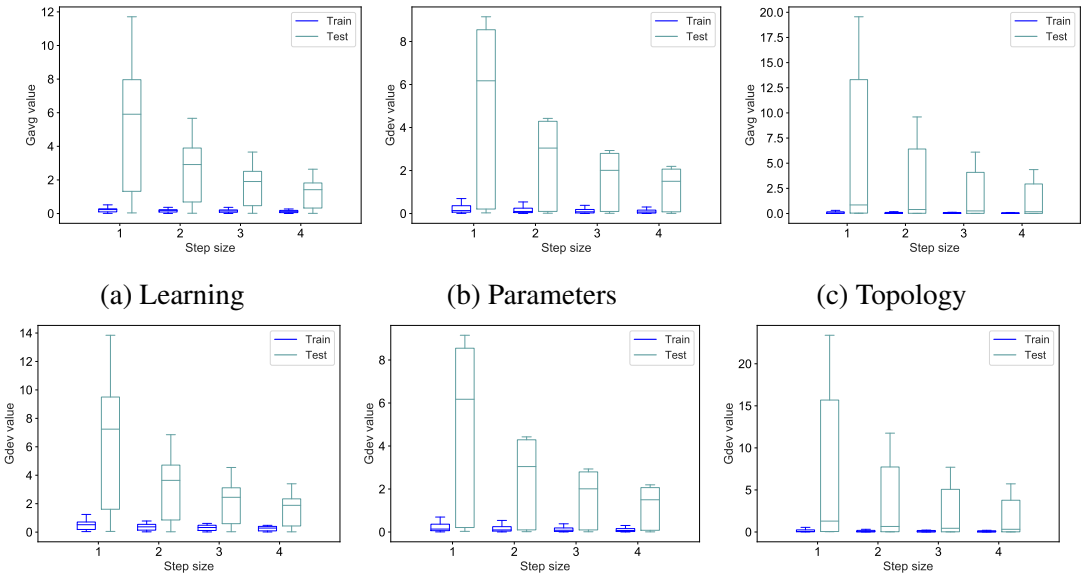


(a) Learning (b) Parameters (c) Topology

(d) Learning (e) Parameters (f) Topology

Figure 4.9: *SVHN dataset*. The organization of the plots is analogous to Figure 4.6.

topology mutations), that ruggedness corresponds to some sudden jumps and not accentuated cliffs/ravines. For SVHN, regarding both learning and parameter mutations, the results are very similar to those of CIFAR10, where we have smooth landscapes with the presence of cliffs/ravines. As for the topology mutation operator, according to the autocorrelation results (Figure 4.4f), it produces rugged landscapes, but according to this gradient analysis, both G_{avg} and G_{dev} are very similar, meaning that these rugged elements are more step-like, which explains the capability of producing some semi-stable



(a) Learning (b) Parameters (c) Topology

(d) Learning (e) Parameters (f) Topology

Figure 4.10: *SM dataset*. The organization of the plots is analogous to Figure 4.6.

low loss runs as shown in Figure 4.4c. Finally, regarding SM, the training results are essentially the same as MNIST, and the test results show a very large variance, indicative of very high cliffs/ravines that are clearly shown in the evolutionary plots (Figures 4.5a to 4.5c).

Fitness Clouds and Density Clouds

We now investigate the ability of fitness clouds and density clouds to predict the difficulty of a problem. Also, in this case, the results will be presented by showing one figure for each test problem. In all these figures, the arrangement of the plots is the same: the first (respectively the third) line of plots contains a visualization of the fitness clouds on the training set (respectively on the test set), and the second (respectively the fourth) line contains density clouds on the training set (respectively on the test set). As in the previous section, each column of plots contains the results for a particular mutation operator. Table 4.1 presents the percentage of points that are below or coincident with the identity like for each problem in training and test, which helps assess the difficulty of the problems.

We begin by studying the results obtained on the MNIST dataset (Figure 4.11). As we can see in plots (a), (b) and (c), regardless of the chosen operator, on the training set the problem is deemed easy by the fitness clouds, as the vast majority of the points of the fitness clouds are below the identity line.

These results are confirmed by the density clouds in plots (d) (e) and (f). For all operators, we observe two distinct clusters, both over the identity line, one on good (low) fitness values and the other on bad (high) fitness values. The cluster of highest density is

Table 4.1: Table containing the % of points from the fitness clouds either below or coincident with the identity line for each of the problems.

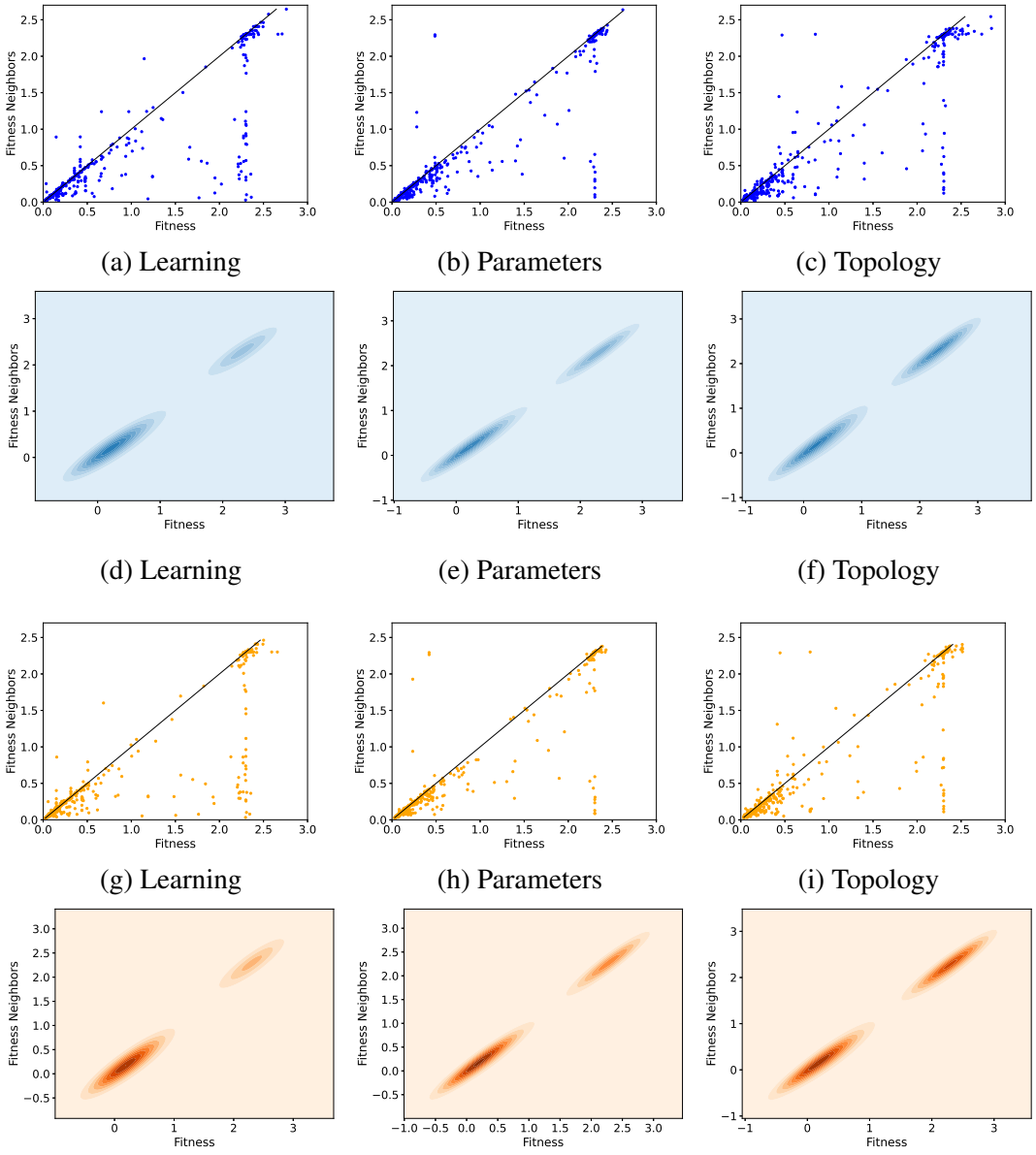
		Learning	Parameter	Topology
MNIST	Train	76	81	81
	Test	75	79	76
FMNIST	Train	83	82	83
	Test	78	79	80
CIFAR10	Train	84	83	84
	Test	82	68	81
SVHN	Train	84	77	84
	Test	72	69	68
SM	Train	77	73	79
	Test	43	39	42

the one closer to the origin, which corroborates the fact that the problem is easy. In fact, the majority of solutions generated by the Metropolis-Hastings sampling are good quality solutions. As for the test set, results are very similar to the ones on the training set. The majority of the points in the fitness clouds are below the identity line, and the highest density clusters in the density clouds are concentrated close the origin. In agreement with the autocorrelation and OFM results discussed previously, fitness clouds and density clouds confirm that neuroevolution has no overfitting for the MNIST dataset.

The results obtained on the FMNIST dataset, reported in Figure 4.12, are identical to the ones of MNIST, with only one small difference concerning the parameters operator: both on the training and on the test set, in the density clouds we can see that both clusters are not completely separate. They are connected since there is a slightly higher density of samples with an average performance.

We now study the results obtained on the CIFAR10 dataset, shown in Figure 4.13. Two different facts can be observed: first of all, in the fitness clouds, the vast majority of the points stands below the identity line; secondly, in the density clouds only one cluster is visible, and it is located over the identity line but rather far from the origin. Combining these two observations, we can conclude that sampling good solutions is hard, even with a sampling method that uses selection pressure like our version of the Metropolis-Hastings. However, the genetic operators are effective, since they are able to improve fitness, often generating better offspring than their parents. Although this is true for all the three studied types of mutation, it is particularly prominent in the learning operator, shown as a protuberance in the density clouds.

The results for the SVHN dataset are reported in Figure 4.14. Again, in the fitness clouds the majority of the points are below the identity line, while in the density clouds only one cluster can be observed, and it is rather far from the origin. However, unlike in the previous problems, the points of the fitness clouds exhibit a very low dispersion, being



(a) Learning (b) Parameters (c) Topology

(d) Learning (e) Parameters (f) Topology

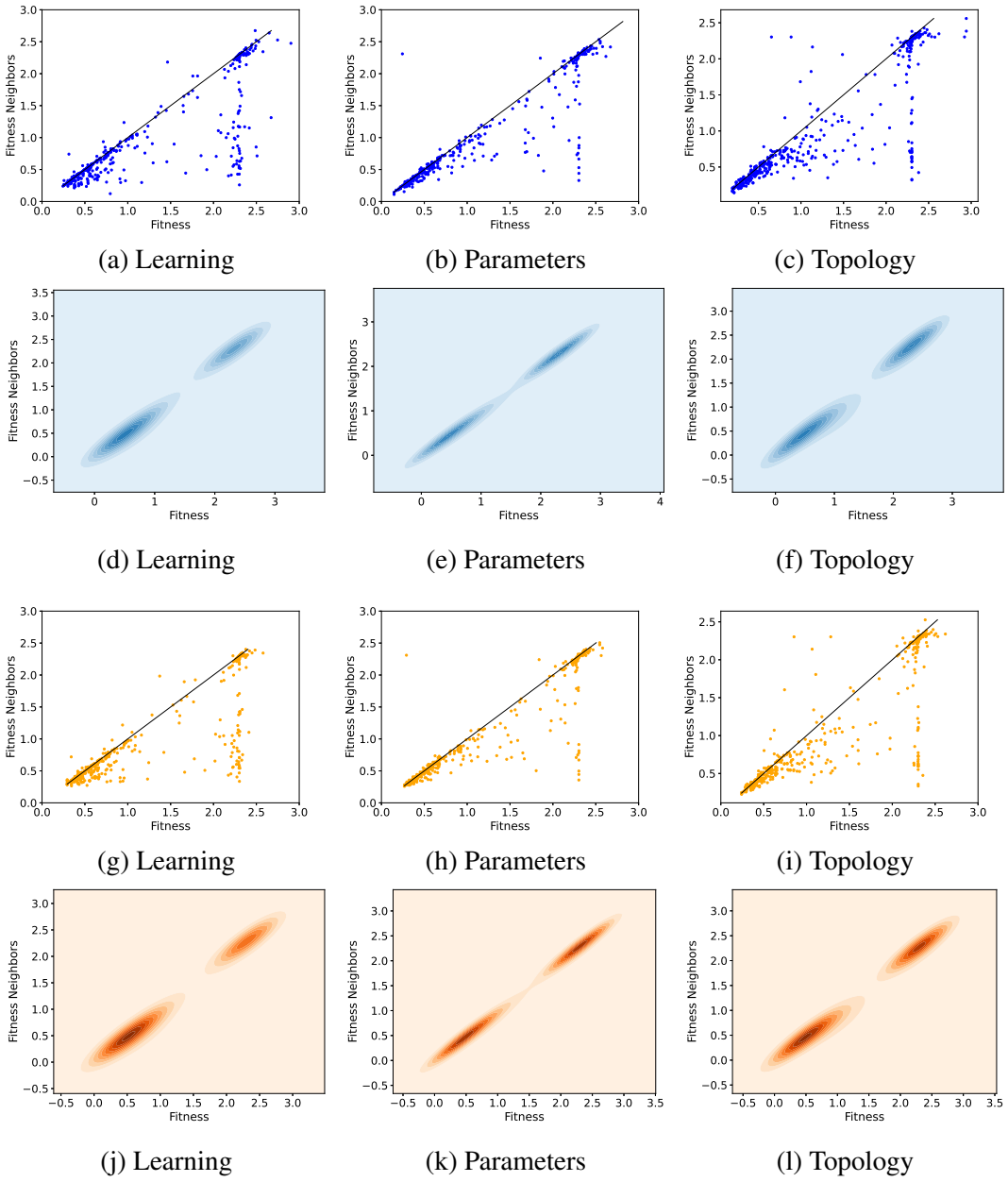
(g) Learning (h) Parameters (i) Topology

(j) Learning (k) Parameters (l) Topology

Figure 4.11: *MNIST dataset*. Plots (a), (b), (c), (g), (h), and (i): fitness clouds; plots (d), (e), (f), (j), (k), and (l): density clouds. Plots (a) to (f) refer to the training set; plots (g) to (l) refer to the test set.

highly concentrated on bad fitness values. Therefore, on one hand, we have an indicator of easiness (points below the identity line) while on the other hand, we have an indicator of hardness (high density of bad fitness values). SVHN was the dataset where the results of the previous section were rather mixed, and indeed it is difficult to draw conclusions from these plots.

The results obtained on the SM dataset are reported in Figure 4.15. On the training set, we can see that the results are similar to the ones obtained on the MNIST dataset: in



(a) Learning (b) Parameters (c) Topology

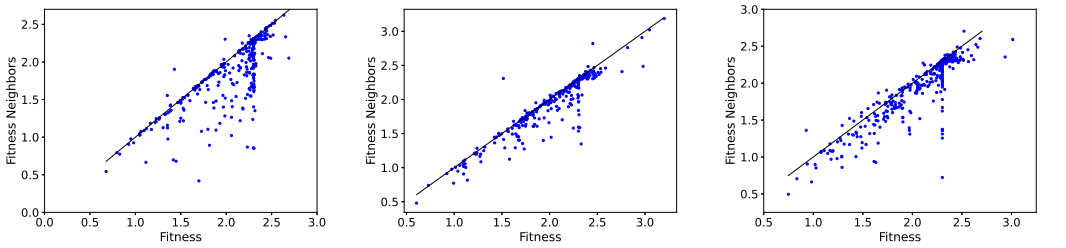
(d) Learning (e) Parameters (f) Topology

(g) Learning (h) Parameters (i) Topology

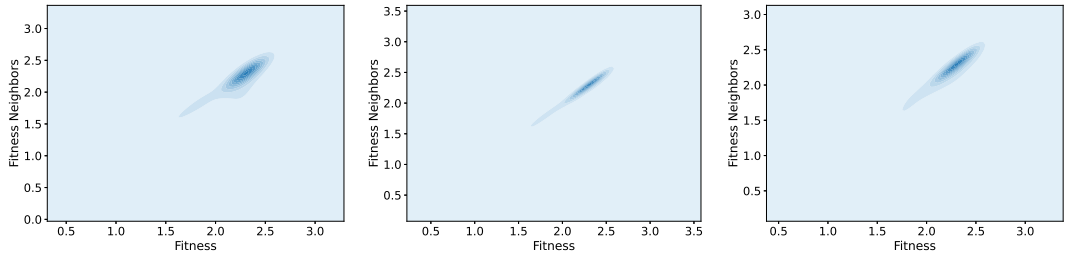
(j) Learning (k) Parameters (l) Topology

Figure 4.12: *FMNIST dataset*. The organization of the plots is analogous to Figure 4.11.

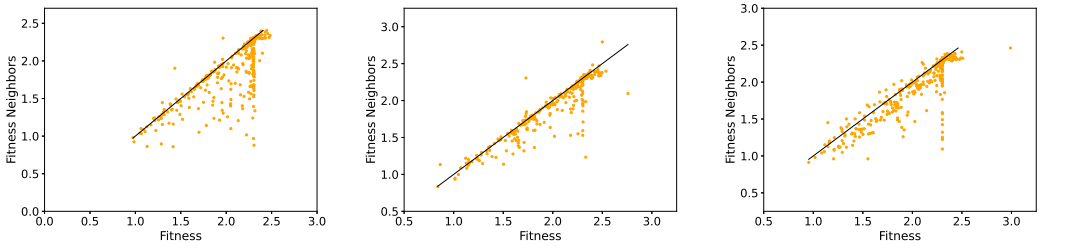
the fitness clouds the vast majority of the points are under the identity line, while in the density clouds we have two clusters, the one with the highest density located close to the origin. But the scenario completely changes on the test set, as expected: in the fitness clouds (plots (g), (h) and (i)), the vast majority of the points are located above the identity line, while in the density clouds (plots (j), (k) and (l)) there is only one visible cluster, and it is located rather far from the origin (notice the different scales between training and test). This remarkable difference between training and test set is a further confirmation of the presence of overfitting, as already observed in the previous section.



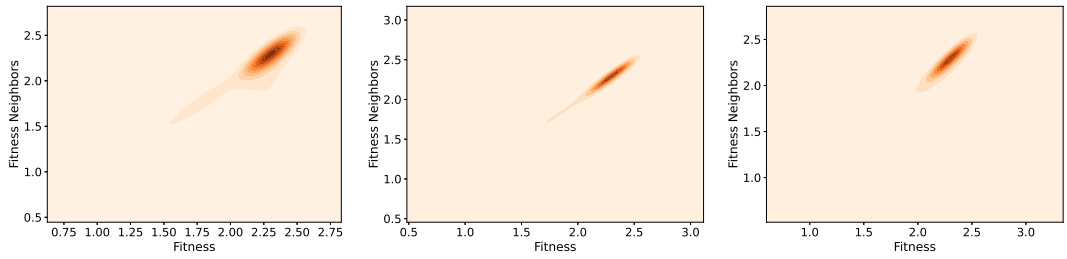
(a) Learning (b) Parameters (c) Topology



(d) Learning (e) Parameters (f) Topology

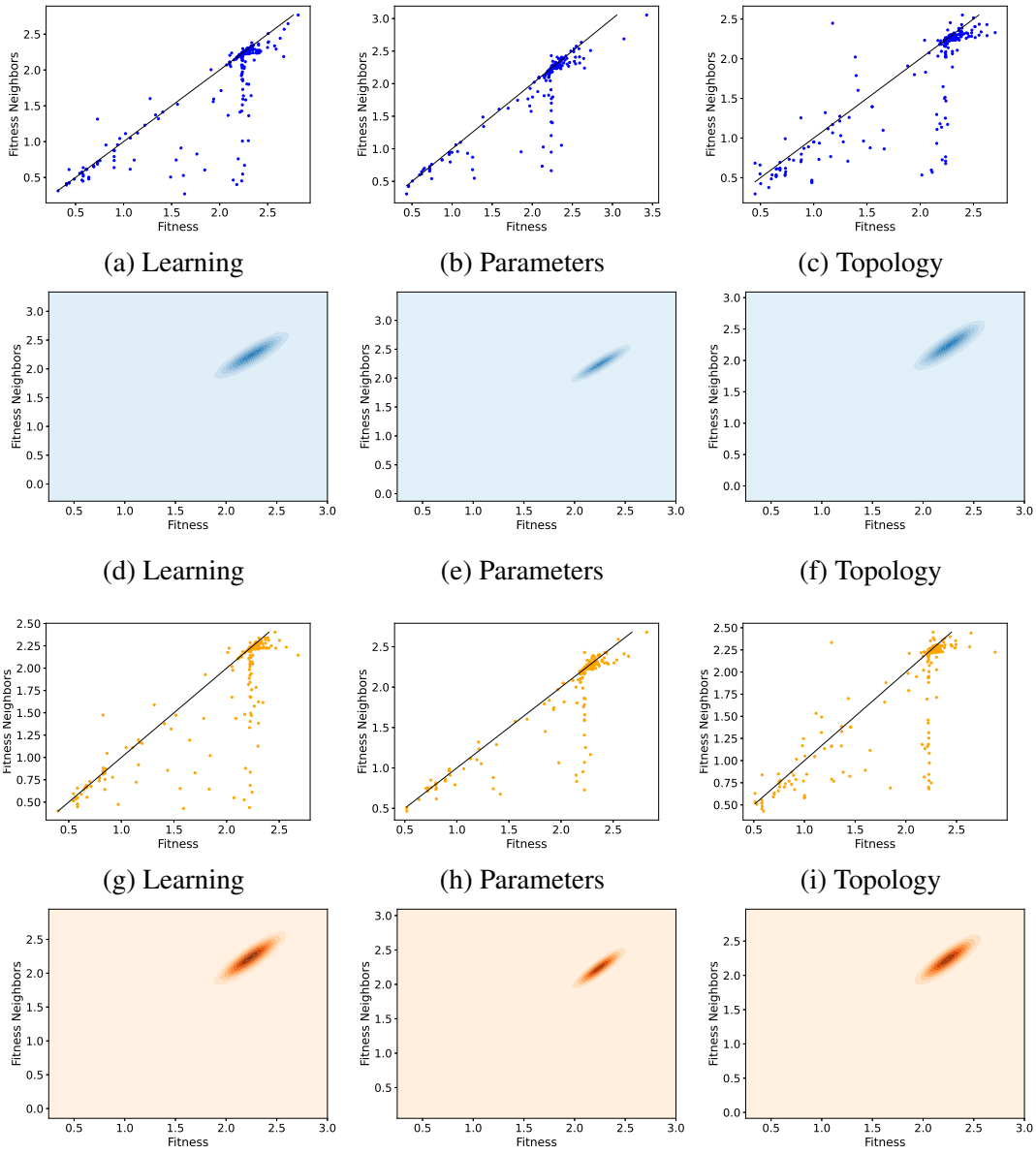


(g) Learning (h) Parameters (i) Topology



(j) Learning (k) Parameters (l) Topology

Figure 4.13: *CIFAR10 dataset*. The organization of the plots is analogous to Figure 4.11.



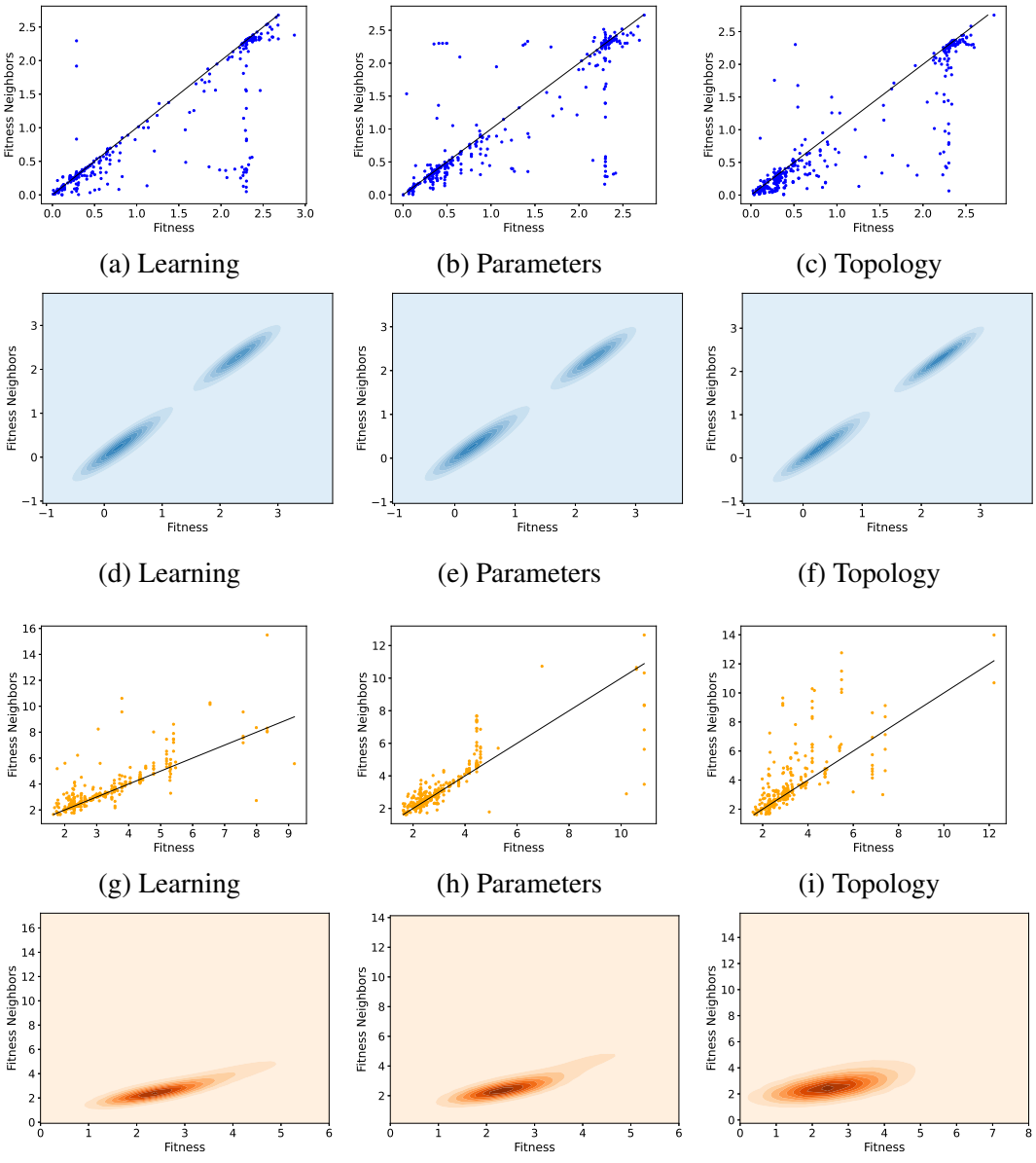
(a) Learning (b) Parameters (c) Topology

(d) Learning (e) Parameters (f) Topology

(g) Learning (h) Parameters (i) Topology

(j) Learning (k) Parameters (l) Topology

Figure 4.14: *SVHN dataset*. The organization of the plots is analogous to Figure 4.11.



(a) Learning (b) Parameters (c) Topology

(d) Learning (e) Parameters (f) Topology

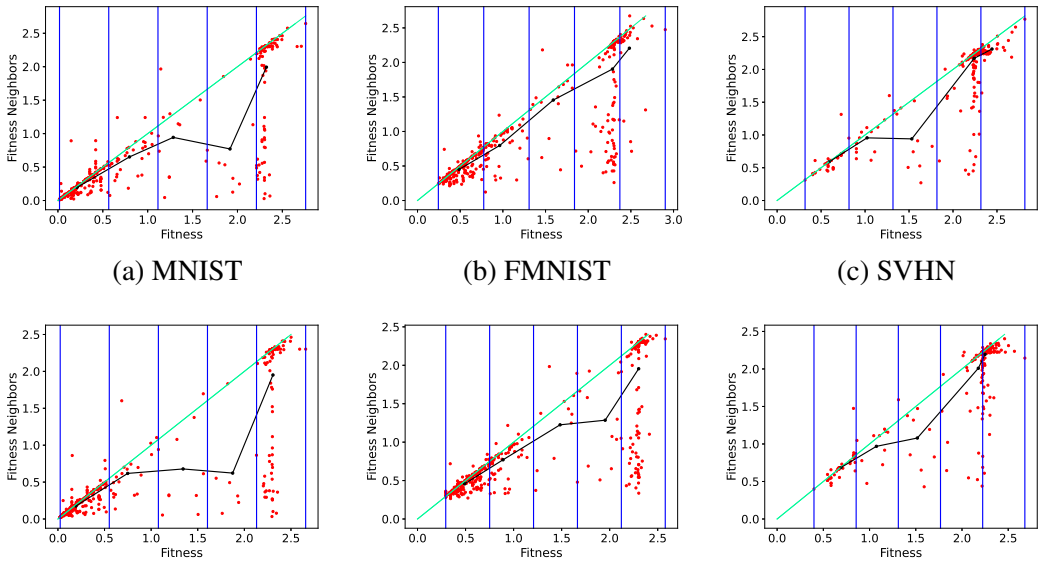
(g) Learning (h) Parameters (i) Topology

(j) Learning (k) Parameters (l) Topology

Figure 4.15: *SM dataset*. The organization of the plots is analogous to Figure 4.11.

Negative Slope Coefficient

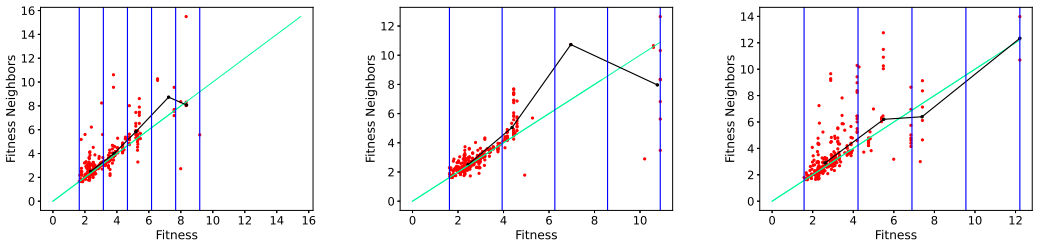
Unfortunately, overall, due to the really large search space, we needed more samples in order to correctly calculate the NSC. As mentioned in 2.4.3, we need to split the fitness cloud into segments to calculate NSC, with a larger number of segments producing a more accurate result. However, as we can see in the previous results, the vast majority of points in the fitness clouds are concentrated in either one or two clusters. This reduces the number of possible segments since with a larger number there would be empty segments. Apart from the short number of segments, there is also the problem of the discrepancy of points in each segment, where we end up having some segments with 200 points and others with ten or eleven. In Figure 4.16 we show some examples of how the sparse point distribution affects this measure. Taking as an example Figure 4.16a, we can see that even only using five segments, the third and fourth segment only contain eleven and ten points respectively, which produces very accentuated slopes that would not be there if we had more samples. Another issue is having a consistent number of segments in each cloud. Looking at Figure 4.16 we can see that we are able to divide the cloud into five different segments, but Figure 4.17 shows that on the test clouds of the SM dataset sometimes it's not possible to have five segments. This lack of consistency produces inaccurate results when making comparisons with different problems, or even within the same problem.



(a) MNIST (b) FMNIST (c) SVHN

(d) MNIST (e) Fashion-MNIST (f) SVHN

Figure 4.16: NSC plots. Plots (a), (b), (c): NSC over the plots of the fitness clouds obtained in the training set using the learning operator; plots (d), (e), (f): NSC over the plots of the fitness clouds obtained in the test set using the learning operator.

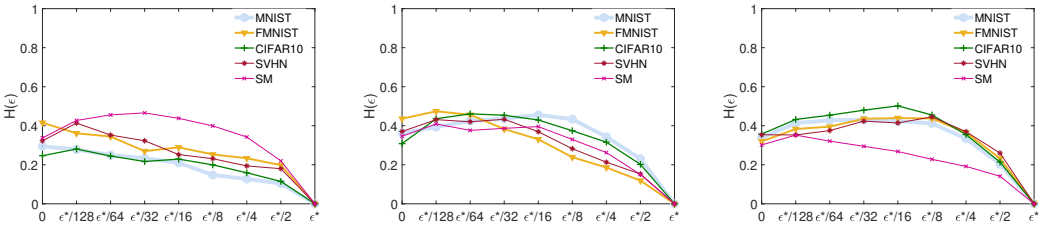


(a) Learning (b) Parameters (c) Topology

Figure 4.17: NSC plots for the test clouds on the SM dataset.

Entropic Measure of Ruggedness

Finally, we study the results of the EMR, reported in Fig. 4.18. Each plot reports the results for one mutation type, showing the values of $H(\varepsilon)$ for multiple ε values (see Sect. 2.4.4) on the five studied datasets. These curves illustrate the trend of how ruggedness changes with respect to neutrality. The results show that, overall, the obtained landscapes have a low degree of neutrality, not maintaining the value of $H(\varepsilon)$ as ε increases. The most neutral landscape is the one produced by topology mutation on the MNIST dataset (plot (c) of Fig. 4.18). Its highest $H(\varepsilon)$ happens when $\varepsilon = \varepsilon^*/64$, but the value suffers minimal change from $\varepsilon = \varepsilon^*/128$ to $\varepsilon = \varepsilon^*/8$.



(a) Learning (b) Parameters (c) Topology

Figure 4.18: Results of the Entropic Measure of Ruggedness $\bar{H}(\varepsilon)$ over different values of ε^* for the the three mutation operators on the four considered test problems.

Table 4.2, which reports the values of R_f for each type of mutation, and for each studied test problem, corroborates the previous discussion: the maximum value for learning mutation is 0.45, while for parameters mutation is 0.47 and for topology mutation is 0.5. Again, we can see that learning mutations induce the smoothest landscapes, while topology mutations induce the most rugged ones. Also, in this case, the prediction of the EMR is compatible with what we observe from the actual neuroevolution runs.

Table 4.2: R_f for each mutation on the studied test problems.

	Learning	**Parameters**	**Topology**
MNIST	0.29	0.45	0.43
FMNIST	0.41	0.47	0.43
CIFAR10	0.28	0.46	0.50
SVHN	0.41	0.43	0.44
SM	0.45	0.40	0.35

Chapter 5

Conclusions and Future Work

In this chapter, we revise the main contributions of this work and the obtained results, and we discuss ideas on how to extend the work in the future.

5.1 Conclusions

The first stage of this work consisted in the development of a grammar-based neuroevolution algorithm for convolutional neural networks, with the encoding choice based on its modularity and flexibility. For this algorithm, due to the vast search space, we chose three different mutation operators that allowed us to study changes in all aspects of the networks. On the next step, we selected eight different fitness landscape analysis measures to characterize the performance of neuroevolution of convolutional neural networks for the first time. The chosen measures were: autocorrelation, overfitting measure, average estimated gradient, standard deviation of gradient, fitness clouds, density clouds, negative slope coefficient and entropic measure of ruggedness. Finally, we proceeded to test our neuroevolution algorithm on four different well studied problems, as well as a hand-tailored version of one of those problems whose objective was to artificially induce overfitting. The results we obtained on these problems confirmed that: all three mutation operators are viable, and each one produces landscapes significantly different from the others; autocorrelation, both gradient measures, fitness clouds and entropic measure of ruggedness were reasonable indicators of problem hardness, both on the training set and on the test set; the overfitting measure was successful in detecting overfitting on both CIFAR and SM problems where it occurred; the density clouds are a very viable alternative to the density of states and deliver a clear representation of the density of points in fitness clouds. The results also showed that the negative slope coefficient measure could not produce accurate results. This was due to the fact that the fitness clouds produced by the three operators had either one or two very prominent clusters, followed by areas with very few points. These areas would then affect the calculation of the slope, resulting in inaccurate results. We think that if it was possible to generate a sample with a much larger

amount of points, this problem could be mitigated and the measure could work.

The majority of these results have already been peer-reviewed and published, and can be considered as an initial step into establishing theoretical foundations for this subfield of evolutionary computation where no previous fitness landscape analysis studies have been done.

5.2 Future Work

Despite serving the purpose of studying the previously mentioned measures, our neuroevolution implementation has some limitations that do not allow the study of other popular measures such as Fitness Distance Correlation or Local Optima Networks. We intent to re-design this implementation in order to overcome these limitations and be able to study the measures we were not able to. For Local Optima Networks, we trust that with some small changes on the grammar combined with significantly more computational power, it would be possible to study this measure. As for Fitness Distance Correlation, it would require some additional changes since we need to find a genotype that encodes a globally optimum solution. This is not something that can be achieved by simply changing the grammar and subsection building rules, and it might require a hand-tailored dataset where we know there is a global optimum that can be achieved. We believe that, by being able to study other measures of fitness landscapes, on additional test problems, we can develop more well established, theoretically motivated predictive tools for neuroevolution, that can significantly simplify the configuration and tuning phases of the algorithm.

Another future task would consist of optimizing data pipelines in order to maximize the efficiency of the algorithms. This would reduce the extremely long training time of the networks, a problem that actually limits the number of usable parameters and thus hinders the quality and accuracy of the results.

We would also like to look further into the results obtained by the topological mutation. Among the studied operators, the topological mutation was the one that returned the poorest results. If these results are originated from the stacking of layers, we hypothesise that they might be due to the degradation of the training error, as proposed in He et al. (2016).

Finally, in a more ambitious note, we would like to see this work expanded to use crossover operators. We idealise that, maybe, if the grammar is restricted in terms of layers and parameters, and the building rules for the subsections are well defined for this goal, it is possible to define a neighborhood function for the crossover, making it possible to apply the studied measures to this operator.

Bibliography

Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *Trans. Neur. Netw.*, 5(1):54–65.

Assunção, F., Lourenço, N., Machado, P., and Ribeiro, B. (2019). Fast denser: Efficient deep neuroevolution. In Sekanina, L., Hu, T., Lourenço, N., Richter, H., and García-Sánchez, P., editors, *Genetic Programming*, pages 197–212, Cham. Springer International Publishing.

Assunção, F., Lourenço, N., Machado, P., and Ribeiro, B. (2018). Denser: deep evolutionary network structured representation. *Genetic Programming and Evolvable Machines*, 20:5–35.

Assunção, F., Lourenço, N., Ribeiro, B., and Machado, P. (2020). Incremental evolution and development of deep artificial neural networks.

Branke, J. (1995). Evolutionary algorithms for neural network design and training. In *Proceedings of the first Nordic workshop on genetic algorithms and its applications*, pages 145–163.

Chen, D., Giles, C. L., Sun, G.-Z., Chen, H. H., Lee, Y. C., and Goudreau, M. W. (1993). Constructive learning of recurrent neural networks. pages 1196–1201 vol.3.

Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection*. Murray, London.

Eiben, A. E. and Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition.

Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62.

Fogel, D. B. (1999). *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*. IEEE Press.

Fogel, L. J. (1962). *Autonomous Automata*, volume 4.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202.

Gallagher, M. (2001). Fitness distance correlation of neural network error surfaces: A scalable, continuous optimization problem. In *Machine Learning: ECML 2001, 12th European Conference on Machine Learning, Freiburg, Germany, September 5-7, 2001, Proceedings*, volume 2167 of *Lecture Notes in Artificial Intelligence*, pages 157–166. Springer.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Gómez, A. B., Sáez, Y., and Viñuela, P. I. (2018). Evolutionary convolutional neural networks: An application to handwriting recognition. *Neurocomputing*, 283:38–52.

Gomez, F., Schmidhuber, J., and Miikkulainen, R. (2008). Accelerated neural evolution through cooperatively coevolved synapses. *J. Mach. Learn. Res.*, 9:937–965.

Gruau, F. (1994). Automatic definition of modular neural networks. *Adapt Behav*, 3(2):151–183.

Gruau, F., Whitley, D., and Pyeatt, L. (1996). A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the 1st Annual Conference on Genetic Programming*, page 81–89, Cambridge, MA, USA. MIT Press.

Gustafson, S. and Vanneschi, L. (2005). Operator-based distance for genetic programming: Subtree crossover distance. In *Proceedings of the 8th European Conference on Genetic Programming*, EuroGP'05, page 178–189, Berlin, Heidelberg. Springer-Verlag.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA.

Jones, T. and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. Technical report, San Francisco, CA, USA.

Kassahun, Y. and Sommer, G. (2005). Efficient reinforcement learning through evolutionary acquisition of neural topologies. In *Proceedings of the European Symposium on Artificial Neural Networks*.

Kimura, M. (1983). *The Neutral Theory of Molecular Evolution*. Cambridge University Press.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization.

Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4.

Koutník, J., Schmidhuber, J., and Gomez, F. (2014). Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, page 541–548, New York, NY, USA. Association for Computing Machinery.

Krizhevsky, A. (2012). Learning multiple layers of features from tiny images. *University of Toronto*.

Langdon, W. B. and Poli, R. (2002). *Foundations of Genetic Programming*. Springer-Verlag, Berlin, Heidelberg.

LeCun, Y. and Bengio, Y. (1998). Convolutional networks for images, speech, and time series. In *The Handbook of Brain Theory and Neural Networks*, page 255–258. MIT Press, Cambridge, MA, USA.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

Livni, R., Shalev-Shwartz, S., and Shamir, O. (2014). On the computational efficiency of training neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, page 855–863, Cambridge, MA, USA. MIT Press.

Lorenzo, P. R., Nalepa, J., Kawulok, M., Ramos, L. S., and Pastor, J. R. (2017). Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, page 481–488, New York, NY, USA. Association for Computing Machinery.

Loshchilov, I. and Hutter, F. (2016). Cma-es for hyperparameter optimization of deep neural networks.

Madras, N. (2002). *Lectures on Monte Carlo Methods*. Fields Institute Monographs 2472-4173. American Mathematical Society.

Malan, K. and Engelbrecht, A. P. (2009). Quantifying ruggedness of continuous landscapes using entropy. pages 1440–1447.

Malan, K. M. and Engelbrecht, A. P. (2013). Ruggedness, funnels and gradients in fitness landscapes and the effect on pso performance. In *2013 IEEE Congress on Evolutionary Computation*, pages 963–970.

Maniezzo, V. (1994). Genetic evolution of the topology and weight distribution of neural networks. *Trans. Neur. Netw.*, 5(1):39–53.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., and Hodjat, B. (2019). Chapter 15 - evolving deep neural networks. In Kozma, R., Alippi, C., Choe, Y., and Morabito, F. C., editors, *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293 – 312. Academic Press.

Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, page 379–384, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Moriarty, D. E. (1998). *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks*. PhD thesis, USA.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.

Ochoa, G., Tomassini, M., Vérel, S., and Darabos, C. (2008). A study of nk landscapes' basins and local optima networks. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, page 555–562, New York, NY, USA. Association for Computing Machinery.

Ochoa, G., Verel, S., and Tomassini, M. (2010). First-improvement vs. best-improvement local optima networks of nk landscapes. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I*, PPSN'10, page 104–113, Berlin, Heidelberg. Springer-Verlag.

Ordóñez, F., Sensors, D. R., and undefined 2016 (2016). Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16:115.

Palmer, R. (1991). *Optimization on rugged landscapes*.

Parra, J., Trujillo, L., and Melin, P. (2014). Hybrid back-propagation training with evolutionary strategies. *Soft Computing*, 18(8):1603–1614.

Rakitianskaia, A. S., Bekker, E. V., Malan, K. M., and Engelbrecht, A. P. (2016). Analysis of error landscapes in multi-layered neural networks for classification. pages 5270–5277.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2902–2911. JMLR.org.

Rodrigues, N., Silva, S., and Vanneschi, L. (2020). A study of fitness landscapes for neuroevolution.

Rodrigues, N. M., Silva, S., and Vanneschi, L. (2020). A study of generalization and fitness landscapes for neuroevolution. *IEEE Access*, 8:108216–108234.

Rosé, H., Ebeling, W., and Asselmeyer, T. (1996). The density of states - a measure of the difficulty of optimisation problems. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, PPSN IV, page 208–217, Berlin, Heidelberg. Springer-Verlag.

Ryan, C., Collins, J. J., and O'Neill, M. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of the First European Workshop on Genetic Programming*, EuroGP '98, page 83–96, Berlin, Heidelberg. Springer-Verlag.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition.

Stadler, P. F. (2002). *Fitness landscapes*, pages 183–204. Springer Berlin Heidelberg, Berlin, Heidelberg.

Stanley, K. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8:131–162.

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life*, 15(2):185–212.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127.

Suganuma, M., Shirakawa, S., and Nagao, T. (2017). A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, page 497–504, New York, NY, USA. Association for Computing Machinery.

Sun, Y., Xue, B., and Zhang, M. (2017). Evolving deep convolutional neural networks for image classification. *ArXiv*, abs/1710.10741.

Tsironi, E., Barros, P., Weber, C., and Wermter, S. (2017). An analysis of convolutional long short-term memory recurrent neural networks for gesture recognition. *Neurocomput.*, 268(C):76–86.

Vanneschi, L. (2004). *Theory and Practice for Efficient Genetic Programming*. PhD thesis, Faculty of Sciences, University of Lausanne, Switzerland.

Vanneschi, L., Castelli, M., and Silva, S. (2010). Measuring bloat, overfitting and functional complexity in genetic programming. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, page 877–884, New York, NY, USA. Association for Computing Machinery.

Vanneschi, L., Clergue, M., Collard, P., Tomassini, M., and Vérel, S. (2004). Fitness clouds and problem hardness in genetic programming. In Deb, K., editor, *Genetic and Evolutionary Computation – GECCO 2004*, pages 690–701, Berlin, Heidelberg. Springer Berlin Heidelberg.

Vanneschi, L., Tomassini, M., Collard, P., and Vérel, S. (2006). Negative slope coefficient: A measure to characterize genetic programming fitness landscapes. In Collet, P., Tomassini, M., Ebner, M., Gustafson, S., and Ekárt, A., editors, *Genetic Programming*, pages 178–189, Berlin, Heidelberg. Springer Berlin Heidelberg.

Vanneschi, L., Tomassini, M., Collard, P., Vérel, S., Pirola, Y., and Mauri, G. (2007). A comprehensive view of fitness landscapes with neutrality and fitness clouds. In Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., and Esparcia-Alcázar, A. I., editors, *Genetic Programming*, pages 241–250. Springer Berlin Heidelberg, Berlin, Heidelberg.

Vassilev, V. K. (2000). *Fitness Landscapes and Search in the Evolutionary Design of Digital Circuits*. PhD thesis, Napier University.

Vassilev, V. K., Fogarty, T. C., and Miller, J. F. (2000). Information characteristics and the structure of landscapes. *Evol. Comput.*, 8(1):31–60.

Vassilev, V. K., Fogarty, T. C., and Miller, J. F. (2003). *Smoothness, Ruggedness and Neutrality of Fitness Landscapes: from Theory to Application*, pages 3–44. Springer Berlin Heidelberg, Berlin, Heidelberg.

Verbancsics, P. and Harguess, J. (2015). Image classification using generative neuro evolution for deep learning. In *Proceedings of the 2015 IEEE Winter Conference on Applications of Computer Vision*, WACV '15, page 488–493, USA. IEEE Computer Society.

Verel, S. (2006). Evolutionary computation and fitness landscapes. *Seminaires doctorants*, 1:5–6.

Verel, S. (2012). Fitness landscapes and graphs: Multimodularity, ruggedness and neutrality. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, page 1013–1034, New York, NY, USA. Association for Computing Machinery.

Verel, S., Collard, P., and Clergue, M. (2003). Where are bottlenecks in nk fitness landscapes? volume 1, pages 273 – 280 Vol.1.

Weinberger, E. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5):325–336.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning.

Wright, S. (1932). *The roles of mutation, inbreeding, crossbreeding, and selection in evolution.*

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Xie, L. and Yuille, A. L. (2017). Genetic cnn. pages 1388–1397.

Xin Yao (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447.

Yao, X. and Liu, Y. (1997). A new evolutionary system for evolving artificial neural networks. *Trans. Neur. Netw.*, 8(3):694–713.

Young, S. R., Rose, D. C., Karnowski, T. P., Lim, S.-H., and Patton, R. M. (2015). Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, MLHPC '15, New York, NY, USA. Association for Computing Machinery.

Yu, F. and Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122.